Snapshots of modern mathematics
from Oberwolfach

№ 1/2016

Swarming robots

Magnus Egerstedt

When lots of robots come together to form shapes, spread in an area, or move in one direction, their motion has to be planned carefully. We discuss how mathematicians devise strategies to help swarms of robots behave like an experienced, coordinated team.

1 Lots of robots

Swarm robotics is the study of how lots of robots can solve tasks together. This is tricky because each robot has to make decisions, but can access only limited information. The individual robot's decisions should result in elegant and useful team-level behaviors, like in schooling fish or swarming insects.

In a number of current and future applications it makes sense to deploy large teams of robots working together. For example, after natural disasters a swarm of robots could search for survivors. Similarly, on the manufacturing floor, on farm fields, or even in space, many robots could work together to cover a wider area, accomplish more tasks, and become more robust to failures: if one robot breaks down, it is no big deal.

We call the desired movement patterns of the robots *swarming behaviors*. When designing them, one first needs a model of what information different robots have access to. Say we have a team of N robots, and we place Robot 1 at position p_1, Robot 2 at position p_2, and so forth. A typical robot can find out where other, near-by robots are, but it cannot determine the position of robots at a large distance. We denote the set of all robots whose position Robot i can determine by N_i, the *neighborhood* of Robot i.

In Figure 1, three robots are drawn as circles and their neighborhood sets indicate that Robots 1 and 3 can only determine where Robot 2 is, while Robot 2 can "see" both Robots 1 and 3.

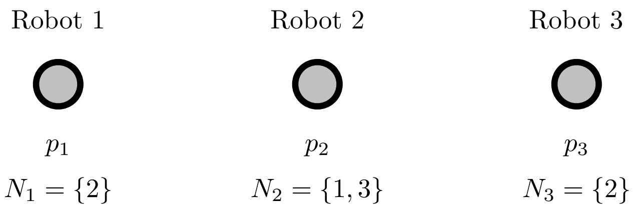


Figure 1: Three robots arranged on a line, such that Robot 2 can see both Robots 1 and 3, but those two robots cannot see each other.

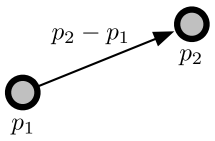
The mathematical question now is how to make the robots move intelligently, given the limited information they have. But such motion algorithms have to follow four very clear guidelines:

1. The algorithms have to be *local* in the sense that each robot can only move based on the information locally available. For example, in Figure 1, Robot 1 can base its decisions on the position of Robot 2, but not on the position of Robot 3.
2. The algorithms have to be *scalable*: when more and more robots join the team, the individual robots should not have to keep track of more and more information, because having to deal with a large amount of complex information could overwhelm them.
3. Whatever the robots do, they have to be *safe*: they are not allowed to collide with each other or with obstacles in the environment.
4. Lastly, the algorithms that describe the individual robot's moves must result in *emergent* (that is, team-level) behaviors. If the robots just end up doing uncoordinated things, then they are not very useful. Instead, we need to know in advance what the team will do.

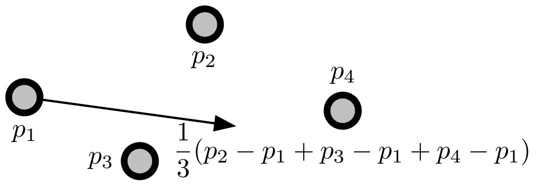
A big part of swarm robotics research is devoted to a question related to the last requirement on the list: how can we ensure that the desired global behaviors emerge from local motion algorithms?

2 Let's meet!

One example of a task swarm robotics can solve is the *rendezvous problem*, where the robots have to move in such a way that they eventually all meet at the same location [1]. But, to make matters worse, the robots are not allowed to



(a) Robot 1 aims towards Robot 2.



(b) The center of N_1 relative to Robot 1 is given by a combination of the vectors from Robot 1 to the other robots.

Figure 2: The rendezvous problem with two and four robots.

communicate or decide in advance where to meet – this has to be decided while the robots are moving around. Moreover, all they can determine is the *relative* position of neighboring robots. That means that even if Robot 1 is a neighbor of Robot 2, Robot 1 cannot determine p_2. Instead, it can only determine $p_2 - p_1$, the position of Robot 2 relative to Robot 1. This is illustrated in Figure 2a, where Robot 1 can determine the vector from itself to Robot 2, given by $p_2 - p_1$.

If Robots 1 and 2 in Figure 2a are to meet, it seems completely reasonable that they should aim towards each other. If we denote the next position of Robot 1 by p_1^{next}, the motion algorithm takes the following form:

$$\begin{aligned} p_1^{\text{next}} &= p_1 + \gamma\,(p_2 - p_1) \\ p_2^{\text{next}} &= p_2 + \gamma\,(p_1 - p_2)\,, \end{aligned}$$

where γ is a positive number that tells us how large of a step the robot takes. If γ is small, the steps are smaller, while with a larger γ the robots move faster.

If we have more than two robots, it makes sense to use a similar strategy by making each robot "aim" towards the center of its neighborhood. For instance, if Robot 1 has three neighbors (Robots 2, 3, and 4), its neighborhood center is given by

$$\frac{1}{3}(p_2 + p_3 + p_4).$$

But this information is not available to Robot 1, since it can only determine relative displacements! Fortunately, we can rewrite this equation as

$$\frac{1}{3}(p_2 + p_3 + p_4) = p_1 + \frac{1}{3}\left((p_2 - p_1) + (p_3 - p_1) + (p_4 - p_1)\right).$$

This is much better, because Robot 1 can indeed determine the expression inside the parenthesis (the neighborhood center relative to Robot 1), as shown in Figure 2b.

In general, the neighborhood center relative to Robot i is given by

$$\frac{1}{|N_i|} \sum_{j \in N_i} (p_j - p_i).$$

Here $|N_i|$ means the number of neighbors in the neighborhood set N_i. The motion algorithm thus becomes

$$p_i^{\text{next}} = p_i + \gamma \sum_{j \in N_i} (p_j - p_i).$$

This equation tells us that the next position of Robot i should be its previous position plus a step (we replaced $1/|N_i|$ with γ as the step size) towards the center of its neighborhood set.

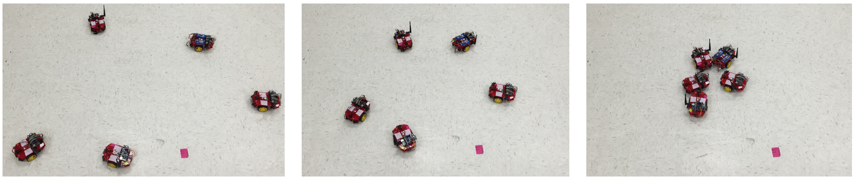


Figure 3: Five robots meet. They all aim towards the centers of their neighborhoods.

It turns out that this simple strategy works [3, 5, 6, 7], as shown in Figure 3, where 5 robots get together. Hence, the algorithm has the desired emergent property. And since the robots only rely on locally available information (the relative position of neighboring robots), the algorithm is local. Moreover, only neighboring robots are taken into account, so the algorithm is indeed also scalable. Sadly, it is certainly not safe, since the robots are actually designed to crash into each other. So, even though this gives us a simple yet powerful swarm algorithm, we need to improve it somehow to enable the robots to execute it in a safe manner.

3 Robots moving together

The main problem with the algorithm that makes the robots meet is that it works too well, with the result that the robots collide. But what if we add a term to the algorithm that ensures the robots never get too close to each other? For example, if there are only two robots in the swarm (as in Figure 2a), we could add a weight to the motion algorithm to stop the robots once the distance between them is small enough. If they are too close, they should instead move away from each other [5].

We use $\|p_1 - p_2\|$ to denote the distance between Robot 1 and Robot 2[1]. If we want this distance to be d, we need a weight function w with the following property:

$$w(\|p_1 - p_2\|) \begin{cases} > 0 & \text{if } \|p_1 - p_2\| > d \\ = 0 & \text{if } \|p_1 - p_2\| = 0 \\ < 0 & \text{if } \|p_1 - p_2\| < d. \end{cases}$$

A simple example of such a weight function is $w(\|p_1 - p_2\|) := \|p_1 - p_2\| - d$, shown in Figure 4.

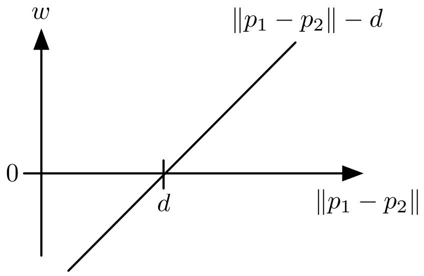


Figure 4: A weight function that helps the robots meet without crashing into each other.

The new, safe algorithm becomes

$$p_1^{\text{next}} = p_1 + \gamma \cdot w\left(\|p_1 - p_2\|\right) \cdot (p_2 - p_1),$$
$$p_2^{\text{next}} = p_2 + \gamma \cdot w\left(\|p_1 - p_2\|\right) \cdot (p_1 - p_2),$$

or more generally,

$$p_i^{\text{next}} = p_i + \gamma \cdot \sum_{j \in N_i} \left[w\left(\|p_i - p_j\|\right) \cdot (p_j - p_i) \right].$$

This type of construction has been used for a lot of different swarming robot applications. The trick then is to find useful weight function [2, 5, 7].

[1] The notation $\|\cdot\|$ is used in mathematics to denote the *norm* or length of something. For the vector $p = \begin{pmatrix} x \\ y \end{pmatrix}$ in the plane, a common norm is $\|p\| := \sqrt{x^2 + y^2}$.

Examples include

- Formation control: form a particular shape
- Coverage control: spread out and cover an area
- Flocking: move in the same direction
- Boundary protection: move along the boundary of a given area
- Multi-robot pursuit: follow each other in some geometrically meaningful manner.

When we solved the rendezvous problem in a safe way, we used our intuition to find a useful weight function. But when the robots have to do something more complex than just meeting in one place, intuition might not suffice anymore. Fortunately, with just a little more mathematics we can generate weight functions in a systematic manner: we define an *energy function* E over the entire team,

$$E := \sum_{i=1}^{N} \sum_{j \in N_i} E_{ij}(\|p_i - p_j\|).$$

The functions E_{ij} indicate how far away $\|p_i - p_j\|$ is from a desired value. They can be imagined as a "cost" or "penalty" that occurs when robots i and j are not at the desired distance from each other. For example, if the goal is to have all robots at a distance d from each other, we can use

$$E_{ij}(\|p_i - p_j\|) := \frac{1}{2}\left(\|p_i - p_j\| - d\right)^2,$$

because this "penalty" is minimal when $\|p_i - p_j\| = d$. Now our goal is to move each Robot in such a way that E decreases. To that end, we calculate, for each robot i, the *gradient of* E *with respect to* p_i, denoted by $\frac{\partial E}{\partial p_i}$. This provides us with a direction with the desired property: the "penalty" E decreases if we move robot i "a little bit" in the direction indicated by the negative gradient. As above, we use γ to denote the size of the step that each robot takes. With this notation, a general movement algorithm is

$$p_i^{\text{next}} = p_i - \gamma \frac{\partial E}{\partial p_i} = p_i - \gamma \sum_{j \in N_i} \frac{\partial E_{ij}}{\partial p_i},$$

where we plugged in the definition of E to obtain the second identity.

This general algorithm has exactly the same form as the aforementioned, safe algorithm. The way one would mathematically prove that this algorithm achieves the desired outcomes is by studying the stability properties of the underlying dynamical system and, in particular, show that the desired configuration is an asymptotically stable equilibrium point to the robot team.

As an example, six robots are using a formation control algorithm to form a circle in Figure 5. To see similar robots moving, take a look at the video [4]. This video was made by Edward Macdonald as part of his Masters thesis on the *assignment problem*, which is the question of what robot should go where in a specific formation.

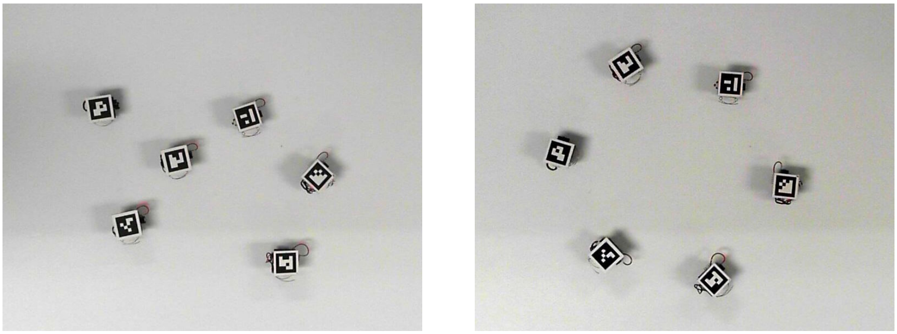


Figure 5: Six robots are forming a circle by executing a formation control strategy.

The study of swarm robotics has only begun and there are a number of things we do not, as of yet, know how to do. In fact, looking at the natural world around us, we are pretty far from making robots as elegant, adaptive, and generally awesome as schooling fish, flocking birds, or swarming insects.

Image credits

All images were created by the author.

References

[1] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita, *Distributed memoryless point convergence algorithm for mobile robots with limited visibility*, IEEE Transactions on Robotics and Automation **15** (1999), no. 5, 818–828.

[2] F. Bullo, J. Cortes, and S. Martinez, *Distributed control of robotic networks. A mathematical approach to motion coordination algorithms*, Princeton University Press, 2009.

[3] 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** (2003), no. 6, 988–1001.

[4] E. Macdonald and M. Egerstedt, *Multi-Robot Assignment and Formation Control*, 2011, https://www.youtube.com/watch?v=se318w2LXD0, visited on December 10, 2015.

[5] M. Mesbahi and M. Egerstedt, *Graph theoretic methods for multiagent networks*, Princeton University Press, 2010.

[6] R. Olfati-Saber, J. A. Fax, and R. M. Murray, *Consensus and cooperation in networked multi-agent systems*, Proceedings of the IEEE **95** (2004), no. 1, 215–233.

[7] W. Ren and R. W. Beard, *Distributed consensus in multi-vehicle cooperative control*, Springer-Verlag, 2008.

Magnus Egerstedt *is a professor of robotics at the Georgia Institute of Technology.*

Mathematical subjects
Numerics and Scientific Computing, Analysis

Connections to other fields
Engineering and Technology, Computer Science

DOI
10.14760/SNAP-2016-001-EN

Junior Editor
Johannes Niediek
junior-editors@mfo.de

Senior Editor
Carla Cederbaum
senior-editor@mfo.de

Mathematisches Forschungsinstitut
Oberwolfach gGmbH
Schwarzwaldstr. 9–11
77709 Oberwolfach
Germany

Director
Gerhard Huisken