Self-Stabilizing Robot Team Formation With Proto

IEEE Self-Adaptive and Self-Organizing Systems 2012 Demo Entry

Jacob Beal
Raytheon BBN Technologies
Cambridge, MA, USA, 02138
Email: jakebeal@bbn.com

Jeffrey Cleveland
Raytheon BBN Technologies
Cambridge, MA, USA, 02138
Email: jcleveland@bbn.com

Kyle Usbeck
Raytheon BBN Technologies
Cambridge, MA, USA, 02138
Email: kusbeck@bbn.com

I. Introduction

We have used the Proto spatial computing language to create teaming algorithms based on random chain formation. Our algorithms are self-stabilizing, scale easily from less than ten robots to thousands of robots, and are highly robust against dynamic changes in perception and communication, arena size, teaming goals, adding and removing robots, and even mobility dimension. In this paper, we describe our approach, give details on our algorithms and their self-* properties, and present simulations validating the algorithms.

The self-organization of robots into teams may be viewed as a spatial computing problem. Spatial computers, as generally defined by the spatial computing research community,[1] are:

> ... collections of local computational devices distributed through a physical space, in which: the difficulty of moving information between any two devices is strongly dependent on the distance between them, and the "functional goals" of the system are generally defined in terms of the system's spatial structure.

In this demonstration we have a collection of robots distributed through geometric space in an "arena." The robots can only perceive and communicate over a strictly limited range (distance dependent information movement) and the goal is to organize the robots into physically separated teams (goals related to spatial structure).

We have thus chosen to build our solution using Proto [1], [2], a spatial computing language that makes it much easier to build self-* distributed algorithms for problems like robot team formation. With Proto, we can program our algorithm not in terms of individual robots, but in terms of aggregates of robots, by viewing the robots as a discrete approximation of the space they occupy. The Proto compiler then transforms the aggregate program into a local program that runs on the individual robots.

II. Teaming Algorithm: Random Chain Formation

Our teaming algorithms are based on random formation of chains. Every robot is either a head or a follower in some chain.

- By default, every robot is the head of a chain, using its own arbitrary unique ID as the ID for that chain and assigning itself an index of 1.
- Heads wander randomly, looking for other chains. If a head encounters a chain with a lower ID, it will follow it backwards towards its tail (the robot with the highest index).
- Tails try to recruit robots from higher ID chains. Recruited robots taken a index one higher than the tail, so a chain of k robots each have a different index in order from head to tail of 1 to k.
- All the robots in the chain try to stay close to the robot in front of them and farther from all other nearby robots.

The chain formation algorithm is made self-stabilizing by having all of the links of the chain actively maintained. This mean that if the chain is ever broken, then either it will be rapidly repaired or else the disconnected robots will form their own chain(s). By modifying the conditions in which a robot will allow itself to be recruited by a tail, various team forming behaviors emerge.

Teams of n Robots: In order to form teams of size n each robot allows itself to be recruited if its position in the recruiting chain will $\leq n$.

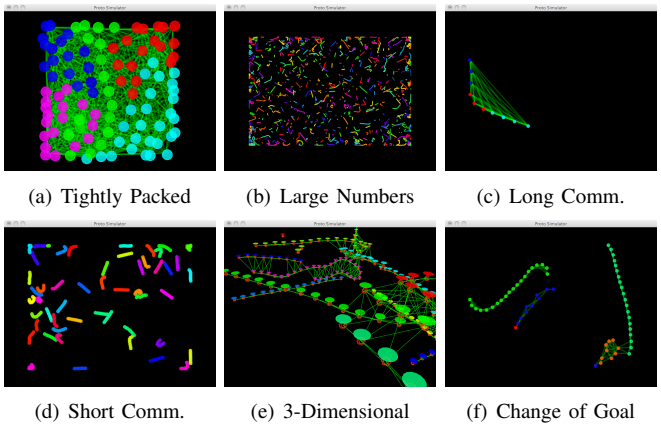
n Teams of Robots: Creating precisely n size-balanced teams of robots is more difficult, because it is impossible to know how big a team should be (and hence whether a robot should be recruited to it or not) without creating a global estimate of the total number of robots or current number of groups.

There is thus a fundamental tradeoff that much be made, between the precision of size balancing and the degree to which robots are allowed to move independently of one another. In one example a single chain of robots is created by specifying that each robot will always allow itself to be recruited. Within that chain each robot assigns itself a team id based on its position in the chain, and how many teams are desired. In the second example, each robot randomly assigns itself a team id between 1 and n. This probabilistic approach, while not as precise as the single chain algorithm, enables the free movement of individual teams.

III. Self-* Properties

Many of the self-* properties of our algorithm stem from the root assumption that the robots are scattered through an

[1]Source: http://scw12.spatial-computing.org, website for the 5th Spatial Computing Workshop.



(a) Tightly Packed (b) Large Numbers (c) Long Comm.

(d) Short Comm. (e) 3-Dimensional (f) Change of Goal

Fig. 1. Algorithm scalability and resilience in changing conditions.

arena of unknown size, with neither global coordinates nor high accuracy movement. This means that (in general) the robots can never safely assume that they have found all of the other robots that are attempting to team with them: a new set of robots with different IDs may appear at any moment, and it is simply that the probability of such encounters drops over time.

An algorithm that can handle this for an arbitrary number of robots is necessarily able to handle many types of self-* challenges as well. Supporting this at a lower level, Proto provides a foundation of an implicitly self-stabilizing view of neighbors' values, self-stabilizing manifold operations (e.g., broadcast), and a domain restriction semantics that ensures clean reinitialization of state in program mode changes. The only other thing we had left to ensure was that the algorithm actively maintained team structure, so that when robots were removed, goals changed, or state was otherwise rendered invalid, the problem could be detected immediately and the incorrect/obsolete state discarded, leaving the teams to rebuild to their new converged state.

With regards to particular self-* properties of interest, this means:

Robustness to perception and communication range: We treat robots as neighbors only when they can both communicate and perceive one another, effectively restricting whichever range is larger to match the smaller. That done, larger ranges simply make it easier for robots to rendezvous with one another, assuming that there is enough bandwidth for them to transmit regularly.

Arena size and complexity scalability: Because robots move randomly, they will eventually rendezvous with one another in an arena of arbitrary size, though the rendezvous time is expected to scale as $\Omega(d^2)$ where d is the diameter of the arena. Non-square arenas and obstacles will not change this significantly so long as the shape is simple.

Number of robots scalability: The only limit on the number of robots is how densely they are packed into the arena: the convergence slows down drastically when robots are packed tightly enough that they have a hard time moving without getting closer than to other robots than their algorithmic limits. We have successfully run the algorithms in simulation with few as two robots [2] and as many as 5000 robots.

Team size/number scalability: Chains, by their nature, can be arbitrarily short or long. Thus, changing the number of teams or the number of robots per team has little effect on the ability of the algorithms to operate correctly. The only limiting factor is that as chains become very large, the random wandering of the head gives more opportunities for a chain to become entangled with itself, which may sometimes result in a team becoming disconnected and having to reconverge.

Scaling from 2D to 3D: Our algorithms are written entirely in terms of vector mathematics, so there is simply no difference between computing with 2D and 3D vectors. The algorithm could scale to higher dimensions as well, were that desired.

Resilience to team specification change: When the specified goal changes between n teams vs. teams of n and/or the value of n changes, then the robots either find themselves in a state where some chain linkages are no longer valid, or else where existing chains must further coalesce. In the first case, since linkages are actively maintained, the obsolete linkages immediately dissolve, leaving robots to reconverge from their newly broken up smaller chains. The second case is no different than if the goal had always been to have larger chains but the robots had not yet managed to converge.

Resilience to perception and communication change: If perception and/or communication shorten faster than the robots can bunch up to compensate, then they may become disconnected and have to reconverge. Otherwise, short ranges simply result in robots moving proportionally slower and closer together to maintain their conncetions. Lengthening ranges have no effect as long as there is enough bandwidth for the robots to transmit regularly.

Resilience to adding robots: Adding robots is no different that encountering robots that had not perviously wandered to a point where they could rendezvous, and thus results in rapid reconvergence.

Resilience to removing robots: Removing robots will often break chains; when this happens, it is no different than a chain being broken due to other reasons (e.g., obstacle avoidance, self-entanglement). Because links are actively maintained, the disconnected portions will rapidly either reconnect or else form their own chains and reconverge.

IV. Algorithm Demonstration in Simulation

We have made a video and all of our simulation code and instructions on executing our work available online at `http://proto.bbn.com/saso2012`

References

[1] J. Beal and J. Bachrach, "Infrastructure for engineered emergence in sensor/actuator networks," *IEEE Intelligent Systems*, vol. 21, pp. 10–19, March/April 2006.

[2] "MIT Proto," software available at `http://proto.bbn.com/`, Retrieved June 22nd, 2012.

[2]The one robot case is trivial.