A novel approach towards computing global maps for multi-robotic operations in tactical environment

Johnson-Bey, Ishmael, Dasari, Venkat

Ishmael Johnson-Bey, Venkat Dasari, "A novel approach towards computing global maps for multi-robotic operations in tactical environment," Proc. SPIE 11419, Disruptive Technologies in Information Sciences IV, 1141908 (22 April 2020); doi: 10.1117/12.2557383

SPIE.

Event: SPIE Defense + Commercial Sensing, 2020, Online Only, California, United States

Invited Paper

A novel approach towards computing global maps for multi-robotic operation in tactical environments

Ishmael Johnson-Bey and Venkat Dasari

CCDC US Army Research Laboratory, Aberdeen Proving Ground, MD, USA

ABSTRACT

During multi-robot simultaneous localization and mapping (SLAM) tasks, a team of robot agents must efficiently synthesize a global map from individual local maps. One method to accomplish this is reserving a subset of robots to perform data fusion and assigning the others to collect and generate local maps. While this solves, the division of labor problem, it requires humans to explicitly designate which robots handle what tasks and it does not scale well to large teams. Moreover, when robots are operating in tactical environments, they may be placed on a heterogeneous team, with limited or unreliable communication infrastructure. To assist with the task of role assignment, we describe a novel decentralized message passing algorithm, for what we are calling Distributed Leader Consensus (DLC). DLC helps a set of agents self-organize into structured groups by giving them the ability to autonomously come to a consensus on the group leader. Our approach is entirely distributed, easily configurable, and is robust to agents being dynamically added to or removed from the system. DLC may be configured to limit group sizes, assign multiple leaders, and select leaders based on computational power and/or physical proximity. We test our approach in simulation by having a set of agents adapt to changing teammate availability.

Keywords: multi-agent systems, decentralized control, multi-agent coordination, distributed consensus

1. INTRODUCTION

Data fusion and near real-time decision-making functions in multi-agent tactical operations require low-latency, non-blocking algorithms for rapid state convergence. Computational cost and resource constraints are also critical factors in the design and development of such algorithms to make them mission deployable. In various domains, teams of robot agents have an advantage over solitary agents due to their ability to complete more tasks in the same amount of time. A single agent must rely on its computational power and hardware affordances, whereas a group can leverage various job division strategies to coordinate task completion. Coordination of multiple agents has been heavily researched and applied to tactical domains such as, terrain exploration, surveillance and reconnaissance, and search and rescue.[1] Each of these domains has a dependency on simultaneous localization and mapping (SLAM) because agents require an understanding of their environment to efficiently complete these tasks. Within Multi-robot SLAM various sub-domains have been studied extensively. These include, data fusion[2–4] and multi-agent coordination.[5–8]

In this paper, we specifically focus on multi-agent coordination while operating in tactical environments. For instance, when military forces are deployed to battlefields, they commonly face the task of having to operate in potentially unknown environments. To reduce human casualties during recon, teams of robots may be deployed ahead of the humans and tasked with mapping the surrounding area. These robot teams have to handle limited resources, unreliable communication infrastructure, and hardware unreliability while still coordinating to compute a global environment map. Therefore, while it may be simple to have all the agents perform the same tasks during exploration, this may not be the best way to allocate their limited computational resources. Especially in the case of heterogeneous robot teams, where some robots may have access to more powerful computing resources or be more suited to a particular task. Proper robot agent coordination is essential for the proper division of labor that efficiently accomplishes the mapping task. Ideally, agents would self-organize into a leadership hierarchy that would enable them to divide tasks and fuse the results of their labor. We propose a message-passing

Send correspondence to Dr. Venkat Dasari: E-mail: venkateswara.r.dasari.civ@mail.mil

Disruptive Technologies in Information Sciences IV, edited by Misty Blowers, Russell D. Hall, Venkateswara R. Dasari, Proc. of SPIE Vol. 11419, 1141908 · © 2020 SPIE
CCC code: 0277-786X/20/$21 · doi: 10.1117/12.2557383

algorithm for what we are calling distributed leader consensus (DLC). DLC is a distributed method by which agents may communicate with one another to autonomously decide on group leaders and divide into groups. Our approach is light-weight, and decentralized, which makes it ideal for scenarios with limited computational resources and unreliable communication connectivity.

For this project, we focus on a scenario in which a heterogeneous squad of robot agents is mobilized to map terrain. Heterogeneity is given by robots having different amounts of computational resources available. Their mission is to navigate and share mapping information to provide rich situational awareness. To conduct the mapping task, agents will self-organize into groups based on the computational resources of their teammates. Leaders should ideally be those with more powerful computing resources. By offloading data fusion to these more powerful agents, we believe that our approach will help to make the process of multi-robot SLAM more efficient for resource-constrained groups. We hope that by properly allocating computational resources among the team, then robot agents will be able to execute more simultaneous tasks or execute more computationally intensive ones.

In the following text, we outline the DLC message passing algorithm and its underlying state machine. In section III we describe the simulation environment we used to test agents running the algorithm. Finally, we conclude by summarizing future next-steps to fully realize our multi-robot SLAM system.

2. DLC ALGORITHM

Here we outline the DLC message passing algorithm which allows a set of agents to self-organize into groups and roles in a distributed and independent manner. Our approach replaces the possibly complex constraint satisfaction calculations[6] with a simple message passing approach that uses a state-machine and leader selection heuristic to dictate agent behavior. Agents send messages to each other across the network, react to types of messages received, change their internal states, and ultimately group together. The agents' state machines operate with four states/roles:

- Solo (see Figure 1)
- Pending Leader (see Figure 2)
- Member (see Figure 3)
- Leader (see Figure 4)

The overall goal of an agent is to either join a group as a "Member" or create one as a "Leader". We consider our agents to be both reactive[9] and holonistic.[10,11] A reactive multi-agent system comprised of two or more agents who's behavior is based around reacting to external events/stimuli. A holonic multi-agent system is made up of agents who group in recursive structures to accomplish a common goal. Holonic systems range in the level of autonomy they provide to agents. Fully autonomous holonic approaches allow agents to pursue personal goals as well as group goals and use a satisfaction function to determine when to trade-off between the two. However, in place of the traditional satisfaction score associated with holonistic systems, our agents judge "satisfaction" by being in one of the grouped states (member or leader). We developed this algorithm to help with dynamic role assignment in multi-robot SLAM scenarios. Our goal is to enable a group of robots to autonomously come to a consensus on which agents perform data fusion and which agents perform data collection. What we have developed is a step toward enabling a group of robots, navigating within tactical terrain, to autonomously organize into groups and produce a shared world-view in a distributed manner.

2.1 Solo State

All agents begin in the Solo state (see Figure 1). The goal of a solo agent is to either become a member group or become a group leader. Before attempting to become a group leader, solo agents will first survey their tactical network for any available groups. It accomplishes this by first broadcasting its information and listening for replies from group leaders that have available space within their groups. We refer to this phase of a solo agent's operation as "Leader Polling". Leader polling lasts for a set amount of time and at its conclusion, a solo agent

will attempt to join a group or transition to the pending leader state. If the agent receives any replies from group leaders, it will pass those leaders' information to the leader selection heuristic. The job of the heuristic is to select the best leader to join based on some set metric. Heuristics may be either a single function or an ensemble that provides a recommendation of which leader to choose from a list of potential leaders. Developers may define different heuristics to optimize selecting leaders based on physical proximity, available compute resources, group size, etc. The result provided by the heuristic will be the leader who is most beneficial to the solo agent. After receiving the recommendation from the heuristic, the solo agent will then send a request to join that leader's group. If they receive back a confirmation message, then the agent changes to the Member state. If not, then, the solo agent will repeat the broadcast-request process for a set number of times before changing themselves to the Pending Leader state and attempting to become a group leader.

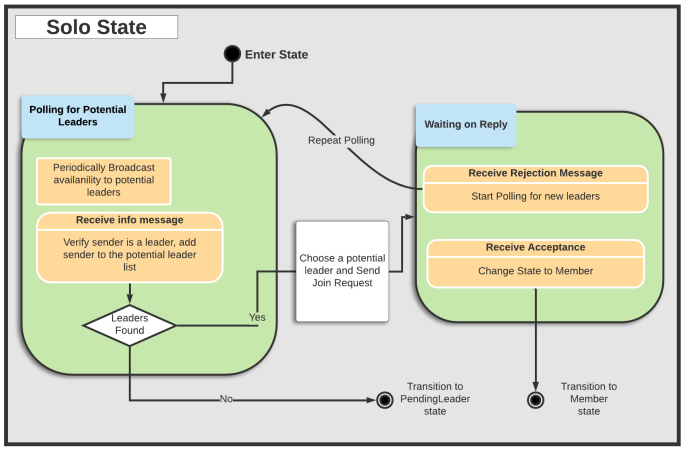

Figure 1. The Solo state. Solo agents seek to join an existing group or become group leaders if no groups are available.

2.2 Pending Leader State

The Pending Leader State (see Figure 2) is the state that all agents must pass though before promoting themselves to be group leaders. The goal of this state is to reduce the likelihood of there being leadership conflicts by providing a probationary period before leadership promotion. Agent's who are pending leadership regularly broadcast "leader check" messages on a set interval. Existing group leaders reply to these messages with conflicts if the maximum number of groups has been established. If the agent does not receive any conflicts during their probationary period then, they will change to the Leader state. However, if they do receive a conflict then, they will try again to assert themselves to the leader state. Pending leaders are given a set number of attempts at becoming a leader. If they reach their max number of attempts at self-promotion, they will demote themselves back to the Solo state and start the process of finding a group again.

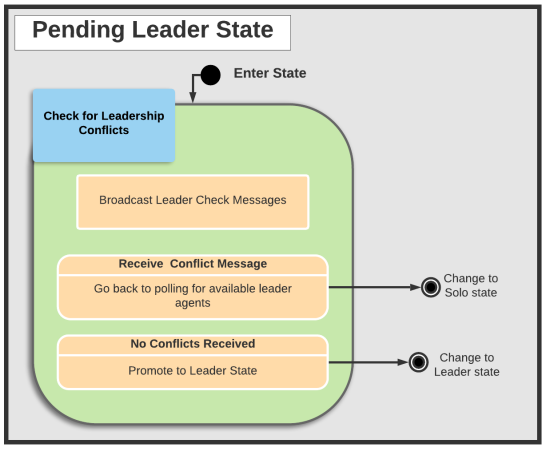

Figure 2. The pending leader state is a temporary state that agents enter prior to self-promoting themselves to be a group leader.

2.3 Member State

The member state (see Figure 3) is one of the goal states for agents within the system. Once in this state, agents are now a part of a group of two or more agents. To prevent their connection to the leader from timing-out, member agents regularly send ping messages to their group leader. In future work, agents in this state will be tasked with exploring their surrounding environment, synthesizing local maps, and sending SLAM information to the leader for data fusion. If at any point a member agent's leader connection times-out, then the agent will return to the Solo state and begin the process of finding a group again.

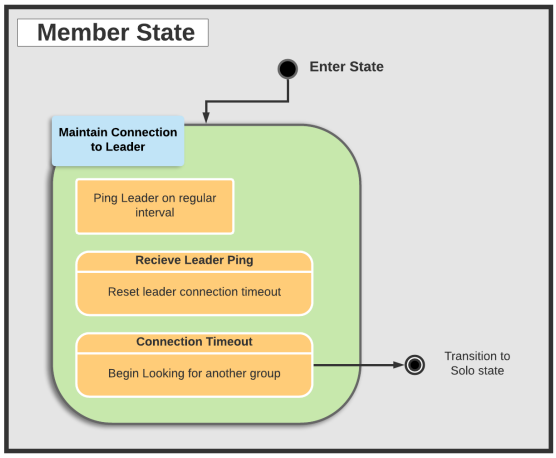

Figure 3. Member agents maintain an active connection to their group leader. If the connection times-out, then they return to the Solo State

2.4 Leader State

The leader state (see Figure 4) is the other goal state for agents within the system. Leader agents are responsible for maintaining group sizes and the maximum number of groups within a tactical network. In the fully-fledged multi-robot SLAM scenario, group leaders would also be responsible for fusing on local maps, received from group members, into a global map to be redistributed to the entire group. Agents will remain in the leader state unless they receive a "conflict" message from another leader indicating that they need to demote themselves back to the solo state. Group leaders play a more reactive role on their team. When they receive information messages from

solo agents, leaders check the size of their group and send back their information if space is available. They then wait for any incoming join requests from these agents. To help restrict the number of leaders on the network, leaders maintain active connections with all other leaders in the system. If another agent attempts to join as a leader, existing leaders will send conflict messages, if too many group leaders are present. Group leaders will also reply to leader-check messages, sent by pending leaders, with conflicts if the max number of leaders are present. During all of these tasks, leader agents also send regularly send out ping messages to all agents in their group and other leaders. As with other states, these ping messages help to prevent connections from timing out. Timed out connections are assumed to be offline agents or agents who are no longer able to communicate due to some external factor.

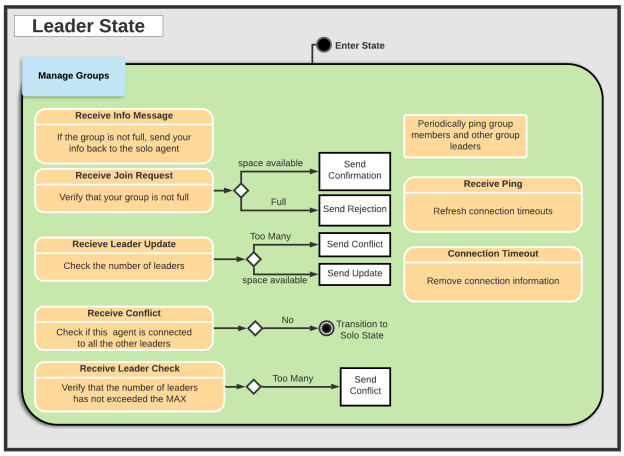

Figure 4. Leaders maintain connections to their group members and other leaders. They are responsible for limiting the number of agents in a group and the total number of leaders within the overall set of agents.

3. METHODS

We implemented the algorithm using Python and the Robot Operating System (ROS).[12] Agents communicate using serialized ROS messages over TCP and multicast networking sockets. To run multiple agents on the same physical host, each agent runs in its own Docker container. We communicate with the ROS master nodes running inside each container using a ROS multi-master setup. From here, we were able to inspect the states off all the virtual agents. We then visualized this information inside of the Unity game engine. Message passing between Unity and the agents is handled by Rosbridge and ROS# by Martin Bischoff at Siemens. We used Unity to manage agent positioning and to visualize the state of agents and the connections between agents **figure 6**. The buildings within the simulation do not yet influence connections between agents. However, we plan to add environmental influence on connections to the simulation.

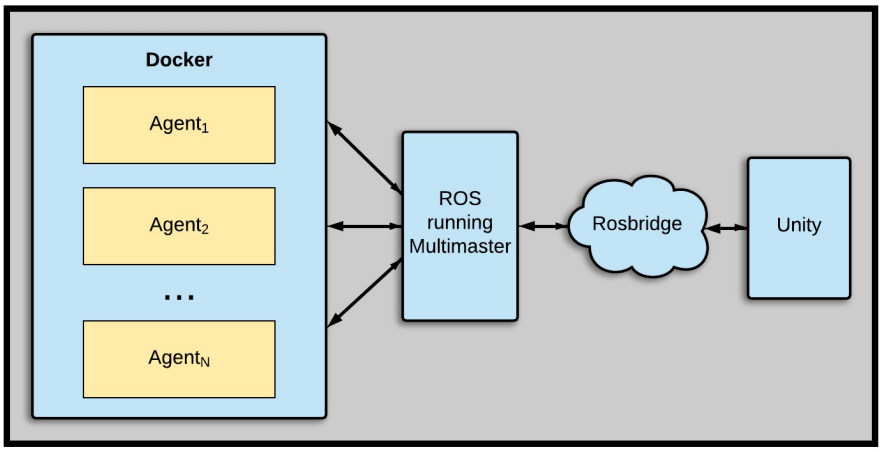

Figure 5. Each agent was run inside its own Docker container. Communication between the agents and the host machine running Unity was handled by a ROS Multimaster node. Communication between the agents and Unity was handled by ROSbridge.

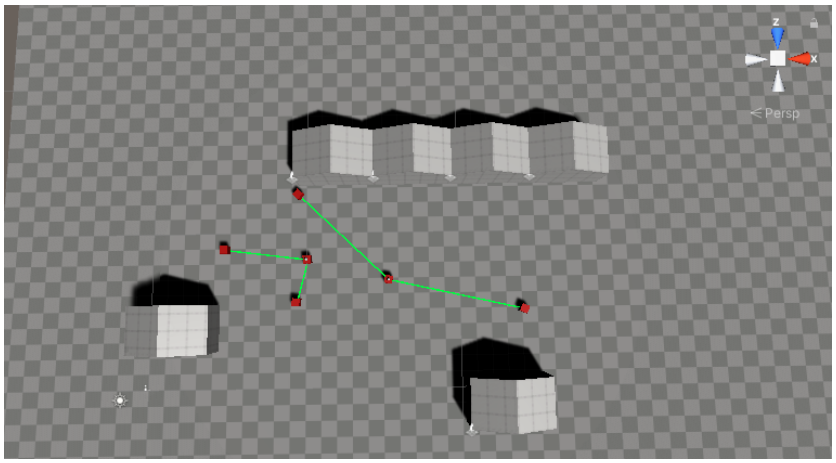

Figure 6. Screenshot of groups of agents operating in the Unity simulation. Agents are represented as red cubes. Connections between member and leader agents are represented by green lines.

4. EXPERIMENTATION AND DISCUSSION

We tested that groups were being formed and that roles remained stable. Agents were brought online at the same time and asked to organize into groups of 3 or less using the shortest distance heuristic for leader selection. During execution, group leaders were periodically switched offline, and their group members had to find new groups. We found that, despite losing their group leader, agents were able to successfully redistribute themselves or form a new group when necessary. Agent behavior also remained stable as the agents were configured to assemble into various other team sizes such as 2, 4, and 5 agents.

During development, we ran into a few design/implementation roadblocks. For example, in an earlier iteration of the system, agents communicated using ROS topics. However, the dynamic creation and removal of ROS topics in the Python API caused unreliable behavior and agents could not maintain groups. Also, we encountered multiple deadlock conditions in which two or more agents simultaneously contested one another for a remaining leader role. This often resulted in two or more agents constantly self-promoting themselves, only to demote themselves once again when the contesting agent tries to obtain leadership. We addressed this problem by

applying a random back-off timer to prevent agents from trying to change their states at the same time. After we applied random back-offs, switched communication to sockets, and limited ROS communication to only ROS services, we had agents that were robust to new agents being dynamically added/removed from the system.

5. CONCLUSIONS AND FUTURE WORK

For this project, we focused on a scenario in which a squad of robot agents is mobilized to map an unknown terrain. To efficiently complete their mission the robots will need to self-organize, navigate, and share mapping information while managing limited resources, unreliable communication infrastructure, and hardware unreliability. To help with organizing under these constraints, we outline the DLC message-passing algorithm which has shown to enable agents to autonomously organize and is robust to agents being added or removed from the system at run-time. Our goal was to provide a framework by which agents can automatically assign roles for data-fusion in Multi-robot SLAM. We also wanted the approach to be light-weight and able to operate in environments with unreliable communications infrastructure. DLC addresses the problem of communication unreliability by being distributed and built around the concept of agents gaining and losing connections between one another. Moreover, we address the complexity of the algorithm by simplifying the distributed decision making into a simple reactionary message-passing system. Finally, the whole algorithm is intended to be easily configurable, to allow for reconfiguration to different tactical environments and scenarios.

In the future, we plan to expand this work by adding additional agent behaviors. For example, giving group leaders the ability to swap roles with one of its group members if that member is more suited to the task of data fusion. Also, we plan to construct a simulation using ROS which demonstrates agents mapping a virtual environment while changing roles and adapting to grouping changes. At the moment, ROS does not have a SLAM stack which supports the dynamic role switching needed for this feat. This remains an active area of development and we hope to see more support as multi-robot scenarios become more popular.

ACKNOWLEDGMENTS

This research was supported by the U.S. Army Research Laboratory (USARL)

REFERENCES

[1] Luo, C., Espinosa, A. P., Pranantha, D., and De Gloria, A., "Multi-robot search and rescue team," in [*2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*], 296–301, IEEE (2011).

[2] Durrant-Whyte, H., Stevens, M., and Nettleton, E., "Data fusion in decentralised sensing networks," in [*Proceedings of the 4th international conference on information fusion*], 302–307 (2001).

[3] Cunningham, A., Paluri, M., and Dellaert, F., "Ddf-sam: Fully distributed slam using constrained factor graphs," in [*2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*], 3025–3030, IEEE (2010).

[4] Cunningham, A., Wurm, K. M., Burgard, W., and Dellaert, F., "Fully distributed scalable smoothing and mapping with robust multi-robot data association," in [*2012 IEEE International Conference on Robotics and Automation*], 1093–1100, IEEE (2012).

[5] de Koning, L., Mendoza, J. P., Veloso, M., and van de Molengraft, R., "Skills, tactics and plays for distributed multi-robot control in adversarial environments," in [*Robot World Cup*], 277–289, Springer (2017).

[6] Xiao, H. and Chen, C., "Leader-follower consensus multi-robot formation control using neurodynamic-optimization-based nonlinear model predictive control," *IEEE Access* **7**, 43581–43590 (2019).

[7] Ye, D., Zhang, M., and Vasilakos, A. V., "A survey of self-organization mechanisms in multiagent systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **47**(3), 441–461 (2016).

[8] King, D. W., Esterle, L., and Peterson, G. L., "Entropy-based team self-organization with signal suppression," in [*The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*], 145–152, MIT Press (2019).

[9] Parunak, H. V. D., ""go to the ant": Engineering principles from natural multi-agent systems," *Annals of Operations Research* **75**, 69–101 (1997).

[10] Rodriguez, S., Hilaire, V., and Koukam, A., "Formal specification of holonic multi-agent systems framework," in [*International Conference on Computational Science*], 719–726, Springer (2005).
[11] Esmaeili, A., Mozayani, N., Motlagh, M. R. J., and Matson, E. T., "A socially-based distributed self-organizing algorithm for holonic multi-agent systems: Case study in a task environment," *Cognitive Systems Research* **43**, 21–44 (2017).
[12] et al., S. A. I. L., "Robotic operating system."