Multi-UGV Multi-Destination Navigation in Coordinate-Free and Localization-Free Wireless Sensor and Actuator Networks

Guyu Zhang, Rastko R. Selmic*, Christian A. Duncan, and Jinko Kanno

Abstract—New Unmanned Ground Vehicle (UGV) navigation algorithms in coordinate-free and localization-free Wireless Sensor and Actuator Networks (WSANs) are presented. The algorithms are distributed and designed for multi-UGV multi-destination navigation in a sensor network environment. The proposed method eliminates possible multiplicity of destination nodes. A Leader Election Algorithm is used to uniquely identify destination nodes, followed by a Hop-distance Assignment Algorithm where each node stores hop-distances from all destinations. The multi-UGV multi-destination navigation planning problem is formulated as a task allocation problem, where each UGV is considered as an agent and each destination as a task. Since even in the special case of a single UGV navigation, the problem, being similar to the well-known Traveling Salesman Problem, is NP-complete, an allocu Zonal and Rastko R. Selmic are suboptimal with respect to the total traveled distance is presented. A communication complexity analysis is provided. Both experimental and simulation results are presented to illustrate the effectiveness of the proposed navigation method.

Index Terms—Unmanned ground vehicle navigation, wireless sensor and actuator networks, task allocation problem.

I. INTRODUCTION

Developments in MEMS-based sensor and actuator technology, low-power RF and A/D design enabled the developments of relatively inexpensive and low-cost wireless sensor and actuator systems [4], [6], [8]. Wireless Sensor and Actuator Networks (WSANs) are commonly used in a variety of applications including home automation, intelligent traffic control, and cyber-physical systems.

As mobile robots become an integral part of human life, we charge them with ever more difficult navigation-related tasks such as search and rescue in remote areas or even planetary explorations. In these tasks, one of the key issues in navigating the robot is to plan the path properly. When several mobile robots are involved, an allocation problem needs to be addressed since proper cooperation among the robots can lead to more efficient task completion.

Centralized approaches maintain global information of the whole system, which can be obtained either from direct connections from a central controller to each individual node, or from indirect connections such as multi-hop links. Centralized designs are prone to single failure points (central controller or algorithm) and usually have a delayed response to local changes. In contrast, distributed algorithms require only local information; they are flexible to changes and robust to individual points’ failures. However, due to the limitation of global information, distributed algorithms usually produce only suboptimal solutions.

Batalin et al. [1], [2] proposed distributed task allocation algorithms using a sensor network. The sensor network is divided into multiple navigation fields based on the priority of tasks that are related to distances from robots to certain tasks. However, some robots might get assigned to the same task if they are originally put in the same navigation field since robots do not participate in the decision making process. When mobile nodes are able to locate themselves in a predefined map, Coltin et al. [7] proposed two algorithms to allocate tasks in wireless sensor networks: an auction-based algorithm and a tree-based algorithm. Results demonstrated that the auction-based algorithm is more efficient regarding the traveling distance while the tree-based algorithm is more efficient regarding the communication cost. Viguria et al. [15] presented an auction-based distributed algorithm that can avoid infinite loops caused by a scenario where two robots share the best bids for at least three tasks. However, this algorithm is not fully distributed since a central robot is needed to control all the bids and assign tasks. Parker [12] introduced a distributed behavior-based task allocation software architecture that is robust and flexible.

A leader election problem is presented for ring networks in [5], [11] and for arbitrary networks in [9], [10]. A good survey of distributed algorithms can be found in [13]. Burns [3] has proved that the lower bound of an asynchronous leader election algorithm is \(\Omega(m + n \log n)\), where \(m\) and \(n\) is the number of links and nodes, respectively. Vasudevan et al. [14] proposed an asynchronous leader election algorithm for dynamic networks, where a source node is responsible for initializing and finalizing the algorithm. However, in our approach, there is no need to find another leader if we can determine the source node first. Single-UGV single-destination navigation is presented in our previous work in [16]. In this paper we consider a more general problem of multi-UGV multi-destination navigation in a WSAN environment, present algorithms in details, and describe both experimental and simulation results.
II. BACKGROUND

Destinations or tasks are defined as different nodes that the UGV needs to visit/service. The problem can be described as a Linear Integer Program. Here we denote:

- the set of \( m \) destinations or tasks as \( \{T_1, T_2, \ldots, T_m\} \)
- the set of \( m \) weights of the tasks as \( \{w_1, w_2, \ldots, w_m\} \)
- the set of \( n \) UGVs as \( \{I_1, I_2, \ldots, I_n\} \)
- the nonnegative utility \( U_{ij} \) of UGV \( I_i \) for task \( T_j \) where \( 1 \leq i \leq n \) and \( 1 \leq j \leq m \).

As is common in real-life scenarios, we assume that \( m \geq n \). An allocation is defined as a set of UGV-task pairs \( (I_i, T_j) \). The task allocation problem is to find an optimal allocation of UGVs to accomplish all tasks (service all destination nodes), i.e., to find nonnegative integers \( \alpha_{ij} \) that maximize

\[
\sum_{i,j} \alpha_{ij} U_{ij} w_j
\]

subject to

\[
\sum_i \alpha_{ij} = 1, \quad 1 \leq j \leq m
\]

\[
\sum_j \alpha_{ij} \geq 1, \quad 1 \leq i \leq n.
\]

Function (1) is the overall system cost, while (2) represents optimization constraints. We define each individual utility as

\[
U_{ij} = \frac{1}{d_{ij}},
\]

where \( d_{ij} \) is the hop distance between UGV \( I_i \) and task \( T_j \). The definition of weight \( w_j \) is application dependent.

III. NAVIGATION ALGORITHMS

The communication in the WSAN is based on an event-driven and message-passing framework. A thread for each node is responsible for handling communication and processing and continuously checks for events in a queue. In particular, if the event is a new message from another node, then the thread reads the new message and processes it. Algorithm 1 shows the framework while individual events and message types are discussed in the following subsections. Note that due to power constraints in WSANs, it is not practical to use a common clock (requires synchronization). Thus, our framework is built on an asynchronous sensor network model.

A. Leader Election Algorithm

Destination nodes (tasks) for UGV navigation need to be identified first. Since multiple sensor nodes can detect a single event simultaneously, a destination is actually composed of a cluster of nodes, all of which detect the same event. Such a cluster of nodes requires a unique identifier. We assume that every node has a unique (ordinal) identifier, and thus we can simply choose the highest identifier among the cluster of nodes. We do this by modifying the standard Leader Election Algorithm where the cluster of nodes votes on a particular node and the highest identifier is declared the leader (see Algorithm 2).

The initiation and execution of this algorithm depends on a few fundamental assumptions. First, we assume that a cluster of nodes associated with a particular destination forms a connected subgraph in the communication network; i.e., nodes can communicate amongst themselves. If this is not the case, such a cluster will be identified as multiple destinations. Second, we assume that the destinations are well separated; i.e., two destinations (clusters of nodes) cannot communicate directly with each other. Otherwise, multiple destinations might be treated as a single destination. Third, we assume that once a node in a cluster detects an event, the other nodes in its cluster will detect the event at relatively the same time. Since the system is asynchronous, there will certainly be differences in timing, but a simple delay in processing the initial passing of messages can accommodate for this difference.

The Leader Election Algorithm begins when a sensor event is triggered. A sensor event is application specific and can be triggered, for instance, when a node receives a dangerously high chemical concentration reading or a high temperature reading or, as in our initial application, when a node determines that it is on the boundary of a hole in the sensing coverage area.

We propose a Timer-based Leader Election Algorithm that instead of transmitting its identification to the entire cluster, sends the \( id \) only to nodes that are at most \( \ell \) hops away, which is initially set to one. In successive passes, as long as the node has not seen a higher identifier, it doubles the distance and retransmits. Once a node has received a higher identifier, the node stops broadcasting its own identifier and simply acts as a relay. Each message transmitted contains two critical components: the potential leader’s \( id \) and the
message lifespan, $\ell_{msg}$, which defines the number of hops the message can be transmitted. Each node keeps track of the current best leader, the highest id known so far, which initially is just that node’s id. When a new message is received, if the id is larger than the current best known, the receiving node updates the maximum value, and if the lifespan is larger than one, with a decreased lifespan by one it retransmits the new identifier. Each pass lasts a certain time $T$, which will be discussed shortly. After the allotted time, triggered by an alarm event, the node retransmits if it still has the highest id seen by it so far. The process terminates once time has passed for a message length that is roughly the diameter of the subgraph (the group of nodes to run the leader election algorithm). This is certainly not more than $n_1$, the number of nodes in the group, but based on the application one could find a tighter estimate. Once the process terminates, the sole leader will broadcast a final message to the cluster to commence the next phase of the algorithm: determining hop distances in the entire network.

When transmitting a message to a node $\ell$ hops away, it is important that the transmitting node waits sufficiently long for the message to propagate through the network. Though it does not cause errors in leader election, a shorter waiting time can increase the chances of more messages being transmitted. We can compute this time delay using $\ell(T_p + T_t)$, where $T_p$ is the time to process a message and $T_t$ is the time to transmit a message. This calculation can easily be adapted to include the time to retransmit in case of errors in communication.

Algorithm 2 presents the Timer-based Leader Election Algorithm in its entirety. Note, procedure PROCESSSENSOREVENT is the procedure initially triggered by a sensing event, procedure PROCESSLEADERALARM is the procedure triggered after every delay, which retransmits the id at progressively longer hop counts, assuming the id is still the maximum seen by that node, and procedures HANDLELEADERELECTIONMESSAGE and HANDLELEADERELECTEDMESSAGE handle the passing of leader messages throughout the cluster.

Before proceeding further, it is important to show that the algorithm does determine a leader for each cluster.

**Lemma 1:** With the maximum waiting time satisfying $T_w \geq d_{max}(T_p + T_t)$, any cluster of nodes will have exactly one elected leader after following Algorithm 2.

**Proof:** Since each node in a cluster is sufficiently close to other nodes in the cluster to communicate amongst themselves but sufficiently separated to not communicate with nodes in any other clusters, we can consider each cluster as a distinct connected graph. Based on the value of $d_{max}$, any message of lifespan $\ell_{msg} \geq d_{max}$ can be received by all other nodes in the cluster. Since $T_w = \ell_{msg}(T_p + T_t)$, the message has sufficient time to reach the furthest nodes. Since each id is unique, this implies that the highest node has sufficient time to transmit its identifier to all other nodes in the cluster. Consequently, since all other nodes in the cluster will at some point have received this identifier, these nodes will no longer consider themselves the leader. 

### Algorithm 2 Timer-based Leader Election Algorithm

1. **procedure** PROCESSSENSOREVENT
2. $\triangleright$ Initialize global variables
3. leaderFlag $\leftarrow$ true $\triangleright$ Node can participate in leader messages
4. $id_{max} \leftarrow id$ $\triangleright$ Current max is node’s id
5. $\ell_{max} \leftarrow \infty$ $\triangleright$ No relay of this value
6. $d_{max} \leftarrow$ estimated diameter of cluster $\triangleright$ Application specific value
7. $\ell_{msg} \leftarrow 1$ $\triangleright$ Initial message lifespan
8. $T_w \leftarrow T_p + T_t$ $\triangleright$ Wait time between retransmissions
9. $id_{l} \leftarrow -1$ $\triangleright$ Stores official leader, once elected
10. $\triangleright$ Start transmitting after slight delay
11. **procedure** PROCESSLEADERALARM($id$)
12. $\triangleright$ Not part of the leader algorithm, ignore message
13. if $id < id_{max}$ then
14. $id_{l} \leftarrow id$ $\triangleright$ This node is the leader
15. broadcast($leaderElected, id_{l}$)
16. **procedure**activateHopCountEvent($id$)
17. $\triangleright$ A new larger id or longer transmission range for max id
18. $id_{max} \leftarrow id_{l}$
19. $\ell_{max} \leftarrow \ell_{r}$
20. broadcast($id_{max}, \ell_{r} - 1$) $\triangleright$ Retransmit with shorter lifespan
21. $T_w \leftarrow 2T_w$
22. $\triangleright$ Increase message lifespan and wait time
23. end if
24. end procedure
25. **procedure** HANDLELEADERELECTIONMESSAGE($id$, $\ell$)
26. if $leaderFlag = true$ then
27. **procedure** activateHopCountEvent($id$)
28. $\triangleright$ First notification of a leader.
29. $id_{l} \leftarrow id$
30. broadcast($leaderElected, id_{l}$)
31. end if
32. end procedure
33. **procedure** HANDLELEADERELECTEDMESSAGE($id$)
34. if $id_{l} = -1$ then
35. $\triangleright$ Not part of the leader algorithm, ignore message
36. $id_{l} \leftarrow id$
37. broadcast($leaderElected, id_{l}$)
38. **procedure** activateHopCountEvent($id$)
39. end if
40. end procedure
41. **procedure** PROCESSALARM($id, \ell$)
42. if $leaderFlag = false$ then
43. return $\triangleright$ Not part of the leader algorithm, ignore message
44. else if $id > id_{max}$ or ($id = id_{max}$ and $\ell > \ell_{max}$) then
45. $\triangleright$ A new larger id or longer transmission range for max id
46. $id_{max} \leftarrow id$
47. $\ell_{max} \leftarrow \ell_{r}$
48. broadcast($id_{max}, \ell_{r} - 1$) $\triangleright$ Retransmit with shorter lifespan
49. end if
50. end procedure
51. **end procedure**

### B. Level Number Assignment Algorithm

Each node calculates and maintains its hop distance to all destination nodes (tasks). The UGVs navigate through the network towards destinations by progressing from one node to a closer-to-destination node. In the proposed solution each node stores a local map $map_{p}$, where the tasks’ ids are set as the map’s keys and the hop distances are set as the map’s values. For simplicity, we assume that the map returns an infinite distance for ids not currently in the data structure.

Algorithm 3 describes the modified Level Assignment Algorithm, which is triggered when the initial cluster nodes determine a winner and invoke the procedure PROCESSACTIVEHOPCOUNT. The other nodes passively process and retransmit level numbers as they are received from other nodes. Since the system is event-driven, there is no need to terminate the process, the messages progress through the network until all nodes have determined their hop distances.
Algorithm 3 Distributed Multi-Destination Level Number Assignment Algorithm

1: procedure PROCESS_TASK
2: \$\textbf{maps}.PUT\{id_s, 0\} \quad \triangleright \ \text{The map} \ \text{is initially empty}
3: \text{BROADCAST}(levelAssignment, \{id_s, 0\})
4: end procedure
5: 
6: procedure HANDLE_LEVEL_ASSIGNMENT_MESSAGE(id_s, ℓ_r)
7: \textbf{if} \textbf{maps}.GET(id_s) > ℓ_r \textbf{then}
8: \textbf{if} \textbf{maps}.GET(id_s) > ℓ_r \textbf{then}
9: \textbf{PUT}(id_s, ℓ_r)
10: \textbf{BROADCAST}(levelAssignment, \{id_s, ℓ_r + 1\})
11: end if
12: end procedure

C. Task Allocation Algorithm

The UGVs passively wait for neighbor sensor nodes to determine hop distances to task clusters. Once these values have been determined, the multiple UGVs must negotiate to determine which task each UGV should tackle. We use the WSAN to store information about the tasks: each node stores an additional local hash map map_c that stores the information on the claiming status of tasks. While the tasks’ ids are set as the map’s keys, each value of the map is an array of length two: the id of the assigned UGV followed by the hop distance between the UGV and the task. Each node’s map_c is dynamic, updating its contents based on new incoming messages. Meanwhile, the UGVs also store a copy of map_c of the same structure constructed during the process. Initially, map_c stores the known distances to each task from the local UGV itself and is updated as new messages arrive.

The algorithm starts when a UGV receives a message that includes the task ids and the hop distances from the tasks. The UGV will then wait a predefined time period to construct its own local hash map map_c. After that, it will choose a task that is the shortest distance away and claim this to the whole network by broadcasting the pair \(T_s, d_h\), where \(T_s\) is the identification of the chosen task and \(d_h\) the distance to this task cluster. In response, nodes that receive this claim will check their local memory to see whether there is another appropriate UGV that has already taken the task. When there is a conflict, such as two UGVs \(I_1\) and \(I_2\) claiming the same task \(T_1\) and their (hop) distances to \(T_1\) are equal, the node will forward the information to the leader node of \(T_1\). The leader will select one UGV by sending a reject message to the other UGVs. For example, if the leader node elects \(I_1\) to fulfill the task, then \(I_2\) will receive a rejection and will need to claim an alternate available task. The pseudocode is shown in Algorithm 4.

Algorithm 4 WSAN-Aided Greedy Task Allocation Algorithm

1: \{Code for the UGVs\}
2: \textbf{variables:} \(id_u\), \(T_s\), \(d_h\) and local map map_c = NULL
3: \textbf{if} receive a level number \(h\) from task \(T_s\) \textbf{then}
4: \textbf{if} map_c.get\{T_s\} == NULL \textbf{then}
5: map_c.add\{(T_s, \{id_u, h\})\}
6: \textbf{end if}
7: \textbf{end if}
8: \textbf{if} find the task \(T_s\) with shortest distance in map_c \textbf{then}
9: \textbf{move} toward \(T_s\) and broadcast \((T_s, \{id_u, h\})\) to the network
10: \textbf{while} receive a message on task \(T_s\) \textbf{do}
11: \textbf{if} local distance is less than received distance \textbf{then}
12: \textbf{update} map_c with the received information
13: \textbf{end if}
14: \textbf{end while}
15: \textbf{if} receive a rejection message \textbf{then}
16: \textbf{find} an alternate task \(T_s\) not claimed by other UGVs in map_c
17: \textbf{move} toward \(T_s\) and broadcast \((T_s, \{id_u, h\})\) to the network
18: \textbf{end if}
19: 
20: \{Code for the nodes\}
21: \textbf{variables:} \(id_s\) and local map map_c = NULL
22: \textbf{while} receive a claim message regarding \(T_s\) \textbf{do}
23: \textbf{if} map_c.get\{T_s\} == NULL \textbf{then}
24: add the received information to map_c
25: broadcast this received information to neighbors
26: \textbf{else} if stored distance is larger than received distance \textbf{then}
27: \textbf{update} map_c with the received information
28: broadcast this received information to neighbors
29: \textbf{else} if stored distance equals received distance \textbf{then}
30: forward tie information to lower level neighbors regarding \(T_s\)
31: \textbf{end if}
32: \textbf{end while}
33: \textbf{if} receive a tied information \textbf{then}
34: \textbf{if} is the leader \textbf{then}
35: \textbf{pick} one UGV and send a rejection message backward
36: \textbf{else}
37: forward the tie information to lower level neighbors regarding certain task
38: \textbf{end if}
39: \textbf{end if}

A. Simulation Results

To test the efficiency of the Timer-based Leader Election Algorithm, we run the algorithm in a randomly generated network where nodes with distinct ids are deployed uniformly in the sensing field.

Network density is calculated as [17]

\[
\sigma = \frac{n_l \cdot \pi r_c^2}{\text{Area}}
\]

with \(n_l\) being the number of nodes and Area being the area of the sensor field (800 \(\times\) 600). A network with 100 nodes was considered. Very large networks were not considered since the leader election algorithm is not expected to run in a large group of nodes. The network density can be adjusted by changing the value of \(r_c\). For example, setting \(r_c = 98\) yields a network density of \(\sigma = \frac{200 \cdot (98^2/\pi)}{600} \approx 6.28\). As Fig. 1 shows, in a uniformly deployed network, the average number of messages transmitted is relatively small. However, the number of transmitted messages is not related to network density.

We also evaluated the proposed WSAN-aided Greedy Task Allocation Algorithm considering only traveling distance without turning angles. A genetic algorithm is used as a
benchmark for comparison purposes. The genetic algorithm tests 80 samples in each iteration and runs 2000 iterations for each simulation. The results obtained by genetic algorithm are roughly considered very close to the optimal solutions. We ran a simple Nearest Neighbor Algorithm where each UGV converges to the nearest destination node that has not been serviced yet without checking if any other UGV is heading towards the same destination.

Fig. 2 shows that the proposed algorithm and nearest neighbor both compare to the genetic algorithm when multiple UGVs start at the same position. We can see that while the proposed algorithm is far better than the nearest neighbor, it’s distance is, in general, not worse than two times that of the genetic algorithm. As shown in Fig. 2, when there are 3 UGVs and 12 destinations, the proposed algorithm’s is around 2.5 times genetic algorithm’s total distance.

Note that no optimal solution can be guaranteed since the system has no global information and coordinates of nodes and UGVs are not available. Therefore, we do not expect that the proposed WSAN-Aided Nearest Neighbor Algorithm has better performance than the Nearest Neighbor Algorithm in each and every case. However, simulation results demonstrate that the proposed algorithm performs better than the simple Nearest Neighbor Algorithm when angle change is also considered.

In Fig. 3 we plot two values: ratio_1 shows the percentage of networks where the proposed algorithm performs better than or equal to the simple nearest neighbor considering distance and angle change, respectively; ratio_2 shows the ratio of cost functions when distance and angle change are both considered. Note that the proposed algorithm outperforms the simple nearest neighbor algorithm with regard to the angle change only (see Fig. 3). In contrast, it performs worse when a large number of targets are present because the network gets dense and turns with large angle changes are needed to navigate among targets.

B. Experimental Testbed and Results

Adepts P3-DX robot is used as a UGV platform. The platform provides reliable movement and is capable of carrying enough load for our experiments.

The Cricket indoor sensor and actuator network system is used as a WSAN platform. The nodes can be configured as beacons or listeners; they are equipped with a paired ultrasound transmitter and receiver, a radio frequency transceiver, a microcontroller and associated peripheral devices. The listeners on the UGV provide a hardware-based implementation
for gradient search algorithms – they listen for lower/higher value of the potential field.

The sensor nodes identify the hop-distance from the target (level-0 node). Listeners on the UGV receive messages from the beacons and let the UGV make decisions about the next navigation step. Fig. 5 shows the trajectory of the UGV in WSAN environment. The network guides the UGV by activating actuator triples, one at a time, in such a manner that hop-distance is reduced. An optimal trajectory is considered as a straight line between the local minimum points in actuator triplets. The ratio of real trajectory length over optimal trajectory length is found to be 1.40. The arrow points to an error that is caused by the erroneous data read from the listeners.

Fig. 5. UGV trajectory in wireless sensor and actuator network.

Fig. 6 shows the potential field values measured as the distance estimate from listeners to the final triplet of nodes. We chose a listeners’ radius $\rho = 20cm$, implying that the local minimum point can be located within 10cm accuracy (see Fig. 4). The decreasing value of the potential field indicates that the UGV is converging towards the destination.

Fig. 6. Potential field measured at listeners during UGV navigation.

V. CONCLUSIONS AND FUTURE WORK

The proposed algorithms are compared with the genetic algorithm in terms of traveled distance and have shown to perform about 2.5 times the genetic algorithm benchmark. Experimental results show that the proposed algorithms are feasible for WSAN implementation and that the obtained UGV trajectory is about 1.40 times the distance of the optimal trajectory. The experiment showed how sensor networks can serve as an effective feedback to mobile robots in the network environment.

Future work will include an energy model for sensor and actuator nodes as well as UGVs that will allow performance evaluation based on energy consumption in addition to traveled distances and/or turning angles.

ACKNOWLEDGMENT

The authors thank Jianfeng Li for experimental implementation and data collection.

REFERENCES