Uncertain Local Leader Selection In Distributed Formations

Dany Rovinsky[1] and Noa Agmon[1]

***Abstract*— Leader-Follower is a hierarchical form of multi-robot formation control, where each robot aims to maintain specific predefined angle and distance from one or more robots in the team (referred to as its local leaders), while a single robot is selected to lead the entire formation to a desired destination. When the robots are given a specific formation to maintain, their goal is usually to minimize the deviation from this desired formation (maximizing the accuracy) during their journey. Previous work has considered optimality in an uncertain environment only in centralized setting (or using perfect, or almost perfect communication). In this paper we examine the problem of optimal multi-robot formation control in a distributed setting, while accounting for two challenges: sensory uncertainty and absence of communication. Specifically, we present an algorithm that allows each individual robot to estimate the overall formation accuracy of the other robots in their field of view via a tree reconstruction algorithm. The algorithm is used to select the most accurate local leader, or to generate virtual local leader via a weighted average of all visible robots. We provide both theoretical analysis and an extensive empirical evaluation (in ROS/Gazebo simulated environment) showing the effectiveness of the two approaches.**

I. Introduction

Moving in a multi-robot formation is a well known problem in robotics. It is typically defined by the need for a team of robots to travel from start point to goal point, while maintaining a specific predefined geometrical pattern. The rational for the pattern is to maximize some team utility function, e.g. increasing fuel economy by reducing the air drag in aircraft formations, maximizing the range of the sensing field for probes or sensor networks, increasing the chances to react to a threat from an unexpected direction of an infantry squad in a patrol or a convoy mission, and more.

A quite simplistic, yet very powerful approach that deals with multi-robot formation task is called leader-follower. In its essence, one robot (the *global leader* or *GL*) is taking care of the navigation to goal and every other robot is following some other team member, which may or may not be the GL. When a robot follows a team member, said team member is called the *local leader* and the following robot a *follower*. This hierarchical approach reduces the formation maintenance down to two tasks: optimal leader selection for each robot (except for the GL), and following a single robot. This paper is focusing on the former.

In most real life applications, fully reliable communication cannot be assured and robots are required to make decisions even in the event of absent communication, for example to avoid putting lives at risk, which is relevant to a variety of applications, from construction to self-driving cars and urban aerial delivery. Therefore, this paper focuses on the *distributed* variant of leader-selection problem, with the assumptions that **(a)** the robots use no explicit communication **(b)** the sensing range of the robots is limited and members of the formation cannot sense every other robot in the team and **(c)** all sensors are noisy and the robots' estimation of their team-members positions are not necessarily accurate. Thus, under the leader-follower approach, chains of followers are created from each robot to the global leader and the positional estimation errors are propagated throughout the chain and might accumulate.

Similarly to the centralized approach in [10] and [11], we present the *Uncertain Leader Selection* (`ULS`) algorithm for *distributed* leader selection that aims to reduce the accumulative error of robots' sensors and increase the formation stability without relying on a non-stable, or even absent communication. This is done using a tree-reconstruction algorithm, where each robot derives the most probable tree representing the hierarchy of local leaders from the observed subset of robots from the formation.

Additionally, considering extremely noisy, or even hostile environments, we present the *Uncertain Virtual Leader Selection* (`UVLS`) algorithm, where instead of following one of the other members of the team, robots follow a virtual point in the formation that they derive from every robot they sense at a given moment. This improves the previous algorithm with regards to robot fault tolerance as well as sensing faults tolerance and uses the redundancy to further reduce individual sensing error. Another difference between the two algorithms is that the `ULS` can be used to select a specific local leader and occasionally reevaluate to confirm this robot is still the best one to follow, while `UVLS` requires continuous execution to assure continuous movement.

In addition to a theoretical analysis that proves the correctness of the methods, we have implemented our algorithms and performed an extensive empirical evaluation in the Gazebo realistic simulation (using ROS). Our empirical results demonstrate the effectiveness of the two methods compared one to another, and compared to a centralized, ground-truth approach, that has full information about the noise levels of each robot's sensors, and use that information to optimally assign leaders for each robot in the formation.

II. Related Work

The problem of formation control is fundamental in the field of multi-robot systems, and as such has received considerable attention in the literature along the years. In this problem, a team of robots should travel in a given shape

*This work was supported in part by ISF grant #1337/15

[1]Dany Rovinsky and Noa Agmon are from the Department of Computer Science, at Bar-Ilan University, Israel. {rovinsd,agmon}@cs.biu.ac.il

throughout an environment while minimizing the difference between the desired shape and the actual one. The methods used for formation control include, among others, behavior-based approaches [2], potential fields [16] and reinforcement learning [20], [5]. The leader-follower approach where each robot maintains a constant separation and/or bearing from other robots, is widely used in the literature especially in distributed systems [2], [7], [11], [9], [10], [3], [18], [8]. The approach is used as a means to reduce the rate of information sharing and communication, reduce computational complexity and support robustness to failures.

One of the main questions in leader-follower formation control is the choice of the local leader. A common representation of the local leader selection is by using a *control graph*, that is, a directed graph representing the assignment of a local leader for each robot in the formation, except for the *global leader* which leads the entire formation [6], [11], [13], [4]. Given the possible sensing capabilities of each robot, and possible cost (and/or uncertainty) of the sensing, one can find a control graph that optimizes a given criterion, for example minimizing the accumulated error, or minimizing local sensing cost of each individual robot [13], [10], [11]. Most leader-follower control algorithms aiming at optimizing stability criteria, to our knowledge, require either reliable communication or centralized processing.

Xu *et al*. [19] and Peng *et al*. [14] examined the leader-follower approach assuming imperfect communication. They assume that there is direct communication between a local leader and its follower, though the communication may be delayed, causing potential divergence of the formation and increasing the deviation from the desired form (error). Therefore, they offer a method for using predictability of the close-horizon to estimate the leader's next steps, allowing it to overcome the communication delays.

Other methods for distributed formation control include the recent work by Alonso-Mora *et al*. [1], that suggest a distributed geometric-based approach for formation control. In their work, they reconfigure the formation in order to optimally avoid collisions with obstacles, by computing the convex hull of the formation. The formation control is distributed, though require communication between team members. In our work the formation is fixed, and no explicit communication is used, and the goal is to keep the formation as close to a given shape (not to reshape it).

Another approach for formation control, which is related to the UVLS algorithm presented in this paper, is the virtual structure approach [12]. In this approach, the entire formation is referred to as one virtual structure, and each robot derives its own trajectory from the knowledge of its relative position in the virtual structure. However, as opposed to the virtual structure approach in which the robots are centrally controlled, or have reliable communication, in our case (the UVLS algorithm) the robots are decentralized, rely only on their local sensing, and do not communicate explicitly with their teammates. Note that the classic virtual-structure approach is a special case of the UVLS where there is perfect full knowledge of all teammates locations at all times.

III. Problem Definition

In this section we will provide some basic graph-theoretical definitions and use them to formally define the formation control problem in our suggested perspective.

All below structures are changing over time. For convenience, we will omit the time component and will address the structures as "visibility graph of r_i", rather than "visibility graph of r_i at time t", when it is clear what the value of t is, or when the reference is relevant for all values of t.

Definition 3.1: The *visibility graph* VG is a digraph whose vertices are the formation robots $\{r_1,\ldots,r_n\}$ and for every $1 \le i,j \le n$ there is an edge from r_i to r_j iff r_i can sense r_j.

Definition 3.2: The *control graph* CG is a tree (the term *control graph* comes from a more general problem, where cycles are allowed and is used here for consistency), whose vertices are the formation robots $\{r_1,\ldots,r_n\}$ and for every $1 \le i,j \le n$ there is an edge from r_i to r_j iff r_j is the local leader of r_i. Motivated by computer vision, this paper's assumes directed sensing, i.e. if r_i and r_j are similarly oriented and r_i can sense r_j, then r_j cannot sense r_i.

Definition 3.3: Two vertices v,u in a tree T are said to be *neighbors* in T if the path between them contains one or no vertices of degree larger than 2. A vertex w with $deg(w) > 2$ that belongs to the path between two neighbors v and u, will be called *the least common ancestor*, or *the lca of v and u*.

Definition 3.4: The *visible set* of a formation by robot r_i is a set $\mathbb{V}_i := \{\, r_j \mid (r_i,r_j) \in E(VG) \,\}$, i.e. it is a set of all robots that are within the sensing range of r_i. Alternatively, it is the set of the robots whose poses are non-infinite in the relevant observation.

Definition 3.5: Given a team of n robots $R = \{r_1,\ldots,r_n\}$, their desired formation is specified by a matrix $F_{orig} = (f_{i,j})$, where for every $1 \le i,j \le n$, $f_{i,j}$ is the pose of r_j in the frame of reference of r_i. For example, consider the diamond formation as in Figure 1:

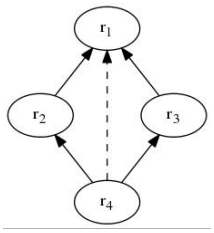


Fig. 1. An example of four robots in a diamond formation with the global leader r_1. An edge between two robots r_i and r_j represents r_j's ability to sense r_i. The dashed edge between r_4 and r_1 represents that r_1 is out of r_4's sensing range.

Assume that the distance between r_4 and r_1 is 4, the distance between r_2 and r_3 is 2, and all robots are facing the same direction. Then the desired formation matrix will be as follows:

$$F_{orig} = \begin{pmatrix} (0,0) & (-1,-2) & (1,-2) & (0,-4) \\ (1,2) & (0,0) & (2,0) & (1,-2) \\ (-1,2) & (-2,0) & (0,0) & (-1,-2) \\ (0,4) & (-1,2) & (1,2) & (0,0) \end{pmatrix}$$

Definition 3.6: For $t \in \{1,2,\ldots\}$, and for every $1 \le i \le n$, an *observation* of robot r_i at time t is $\widehat{O}_i^t = < \overline{p}_{i,1}^t, \ldots, \overline{p}_{i,n}^t >$

where $\overline{p}^t_{i,j}$ is the estimated pose of r_j by r_i. The poses are relative to r_i (so $\overline{p}^t_{i,i} = 0 \ \forall t$) and contain errors, i.e. $\overline{p}^t_{i,j} = p^t_{i,j} + Err^t_{i,j}$, where $p^t_{i,j}$ is the true pose of r_j relative to r_i at time t, and $Err^t_{i,j}$ is a random variable in t, representing the measurement error of r_i w.r.t. r_j.

Definition 3.7: In order to measure the overall formation accuracy, we define the *formation deviation vector* $\overline{\Delta}_t = (\overline{\delta_1}, \ldots, \overline{\delta_n})$ such that for every $1 \leq i \leq n$, $\overline{\delta}_i$ is the difference between the true location of r_i and its expected location originated from F_{orig} in the frame of reference of the *GL* at time t. This value will be measured externally during experiments and will not be accessible to the robots.

We use the norm of $\overline{\Delta}$ to compare between the accuracy of the formation at times t_1 and t_2, i.e. we compare $|| \overline{\Delta}_{t_1} ||$ with $|| \overline{\Delta}_{t_2} ||$. The rational behind this, is the hierarchical nature of leader-follower formation control, where each robot is trying to maintain a pose relative to their local-leaders, which in turn are trying to maintain a pose relative to their local-leaders and so on. Implicitly, all followers are trying to maintain a pose relative to the global-leader, and their overall success as a team is captured in the norm of $\overline{\Delta}$.

The *uncertain leader selection* problem can now be defined as follows: Given a desired formation matrix F_{orig} each robot must choose a local-leader from the set of its visible robots at times $t \in \{1, 2, \ldots\}$, that would minimize $|| \overline{\Delta}_t ||$.

The *uncertain virtual leader selection* problem is defined in similar terms, however this time the goal for each robot in the formation is to choose a desired location to advance to, such that reaching this location would minimize $|| \overline{\Delta}_t ||$.

Our proposed solutions are called `ULS` and `UVLS` respectively and rely on the `TreeReconstruction` algorithm to aid each robot to deduce a subtree of the control graph at time t, from the deviation of the visible robots locations as observed at time t and their expected locations as captured in F_{orig} . This algorithm is presented in the next section.

`ULS` can be used to update the local leader selection in real time, as well as infrequently, when the robot follows the selected local leader and updates its selection once in a while, or on demand (as a reaction to changes in the environment).

`UVLS` on the other hand, is highly coupled with the formation control decision mechanism: the robot must move towards the desired location, while constantly updating that point in space, otherwise it will stop. However, the fact that it considers the positions of every visible robot, allows it to be more tolerant to regular, and even byzantine faults of some team members. Additionally, as mentioned previously, it outperforms `ULS` when the estimation errors are not correlated and partially cancel each other out.

IV. Tree Reconstruction Algorithm

In this section we present the `TreeReconstruction` algorithm, which is used as the building blocks for the `ULS` and `UVLS` algorithms. The connection between the tree reconstruction problem and the uncertain leader or virtual leader selection problems is shown in Subsection IV-C.

The `TreeReconstruction` algorithm, presented in this paper, is originated in the `NeighborJoining` algorithm by Saitou and Nei [15], which was later revised by Studier *et al.* [17]. Given a set of leaf vertices in a tree, `TreeReconstruction` recovers that tree. `TreeReconstruction` is, therefore, a generalization of the `NeighborJoining` algorithm, and guarantees reconstruction of *general* n-ary trees (not only binary trees), without adding more time or space complexity to the computation. More formally: Let T be a tree, and denote by $L(T)$ the set of its leaves, then given a subset of vertices $U \subset V(T)$, such that $L(T) \subset U$, and a distance matrix M of vertices from U (here, distance between two vertices is the length of their path), `TreeReconstruction` recovers T.

The resulted tree uses edge weights to mask trivial vertices (i.e. vertices with degree 2), so when an edge between some vertices $v, u \in V(T)$ has weight a, this means there are $a-1$ vertices in the path between v and u, denoted by P_{vu}, such that $\forall w \in P_{vu} \setminus \{v, u\}, deg(w) = 2$. All paths that contain only trivial vertices are condensed to be a single edge, which reduces the space complexity of this tree's representation, while maintaining all features of the tree and allowing simple conversion from the dense representation to the full one. Without using such representation, all space and runtime complexity would depend on the values of the distance matrix, e.g. a tree with n leafs with max distance j, would have $O(nj)$ number of vertices, meaning it will require $O(nj)$ space to represent the tree and any graph algorithm applied to this tree will depend on j in the runtime complexity. With dense representation, the number of vertices is $O(n)$ (See IV-B for full complexity analysis).

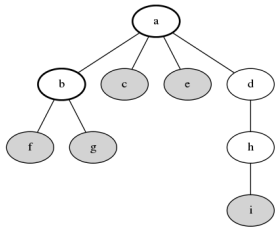


Fig. 2. To demonstrate the execution of `TreeReconstruction` consider the following tree. Here, c, e, f, g and i are the leaves, and a, b are non-trivial vertices which are going to be recovered. The neighbors in this tree are f, g and any pair from $\{i, c, e\}$. The algorithm (given the leaves and their tree distance matrix as the inputs) will first use f and g to recover their *lca* b. Second, it will remove f and h, so b will become the new leaf. Then, c and e will be used to recover a and removed. Next, a and c will "recover" a, which will result in recovering the edge between them. The same will repeat with a and e, and as the final step, with a and i (recovering edge with weight 3).

The helper routines that are used in the algorithm are:

- `find_neighbors` finds the indices of two vertices to be joined, using the *distance_matrix* to derive the matrix of *Q-values*; $Q_{i,j} = (n-2) * d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(k,j)$. The pair i, j with the minimal value guarantees i and j are neighbors, as detailed in [17].
- `update_tree` identifies the *lca* of the given vertices. This method is explained in depth below (see alg 2).
- `update_distance_matrix` deletes the rows/columns of the two given vertices from the distance matrix, and

adds the row and column of the found *lca*.

- `update_list` removes *vertex_1* from the *list* and replaces the one at *index_2* with the *new_vertex*.
- `find_root` uses heuristic to identify the root vertex (representing GL) from the reconstructed unrooted tree.

Algorithm 1 `TreeReconstruction` (accepts U and M)

$T \leftarrow$ INIT_DISCONNECTED_TREE(U)
while $U.size() > 2$ **do**
  $x, y \leftarrow$ FIND_NEIGHBORS(M)
  $a \leftarrow$ FIND_WITNESS(M, x, y)
  $d \leftarrow M[x][y]$
  if $(M[x][a] < M[y][a])$ **then**
    $x, y \leftarrow y, x$
  end if
  $\delta \leftarrow M[x][a] - M[y][a]$
  $v \leftarrow$ UPDATE_TREE$(T, U[x], U[y], d, \delta)$
  UPDATE_DISTANCE_MATRIX(M, x, y, d, δ)
  UPDATE_LIST$(U, U[x], U[y], v)$
end while
UPDATE_TREE$(T, U[0], U[1], M[0][1], M[1][0])$
root $\leftarrow$ FIND_ROOT$()$
return T, *root*

Similar to the original `NeighborJoining` algorithm, `TreeReconstruction` finds a pair $u, v \in U$, such that u and v are neighbors in T, recovers their *lca* , and connects the neighbor vertices to that *lca* . The main differences between the two algorithms are:

- No requirement from the tree to be binary. If two neighbors are identified and it turns out one of them is an ancestor of the other, they will be connected with an edge, rather than via an intermediate vertex.
- Agglomerative vs divisive approach: Rather than begin with all vertices connected to some central vertex and split/merge existing edges, as more vertices are discovered, `TreeReconstruction` begins fully disconnected and adds edges while discovering new vertices. This aids with dealing with non-binary trees, as the divisive approach requires handling disconnected components or cycles that may arise when two existing vertices are connected without a proxy in a star tree.

The aforementioned differences between the two algorithms are demonstrated in the `update_tree` function, used in algorithm 1:

Algorithm 2 `update_tree`(accepts T, x, y, d, and δ)

if $\delta == d$ **then**
  T.ADD_EDGE(y, x, d) ▷ x is the ancestor of y.
  return y
end if
$v \leftarrow$ T.ADD_NEW_VERTEX$()$
T.ADD_EDGE$(x, v, \frac{d+\delta}{2})$
T.ADD_EDGE$(y, v, \frac{d-\delta}{2})$
return v

Given two vertices x and y, their distance in the tree $d = dist(x, y)$, and the difference of distances from them to any other vertex δ (i.e. $\forall c \in V(T), \delta = dist(x, c) - dist(y, c)$), this routine will add the *lca* of these vertices and edges between them, with edge weight derived from d and δ. The condition above (that $dist(x, c) - dist(y, c)$ is a constant value for all vertices $c \notin \{x, y\}$) is actually both necessary and sufficient for x and y to be neighbors, since it essentially means that there is a single vertex that is part of the paths P_{xy}, P_{xc} and P_{yc} for all vertices c. This fact is used to find the "witness" vertex, i.e. some c to calculate distances from. In case of `TreeReconstruction` , c was chosen as the first vertex (by index) such that it is not x nor y.

In case d equals δ, it is easy to see that one of them is the ancestor of the other and this routine (as opposed to its original analogue) adds an edge connecting the two vertices to each other, rather than creating a new vertex.

A. Correctness of `TreeReconstruction`

Theorem 4.1: Given a tree T and a subset of its vertices U, such that $U \subset L(T)$, the algorithm reconstructs T.

The proof of this theorem is omitted, due to lack of space and can be found in the full version[1]. In short, we prove that given two neighbor vertices (that are discovered identically to the `NeighborJoining` algorithm), the algorithm correctly identifies their lca, and use induction over the number of leaves to prove the algorithm's correctness for any tree.

B. Complexity Analysis

The detailed complexity analysis is also demonstrated in the full version of the paper[1]. For the scope of this paper, we will note that the space complexity is $O(n^2)$, due to the distance matrices and the fact that the number of the discovered vertices cannot exceed the number of leaves. The time complexity is $O(n^3)$, similarly to the original algorithm, since the algorithm constructs a helper $n \times n$ matrix in the `find_neighbors` routine, which is executed at each of the $O(n)$ iterations.

C. From observations to distances

As discussed above, in order to successfully identify the subtree of the control graph, we must find the distances of control graph's paths between each pair of the visible robots. Recall that each robot r_i in the formation receives an *observation at time t*, $\widehat{O}_i^t = < \overline{p}_{i,1}^t, \ldots, \overline{p}_{i,n}^t >$.

From $\widehat{O}_i^t$, each robot will derive the *visible matrix of r_i at time t:* $V_i^t = (v_{q,r}^t)$, where for every $1 \leq q, r \leq k$, and for every $t \in \{1, 2, \ldots\}$, $v_{q,r}^t$ is the position of r_q relative to r_p, from the point of view of r_i at time t.

From the visibility matrix, the robots will derive the *deviation matrix of r_i at time t*, i.e. $\Delta_i^t = (\delta)_{r,q}^t$ to be the matrix of differences between the observed relative pose of $v_{r,q}$ and the expected relative pose $f_{r,q}$, i.e. $\delta_{r,q}^t = ||v_{r,q}^t - f_{r,q}||$.

Finally, each robot will aggregate the *moving average deviation matrix AD* , where each element $\delta_{i,j}$ is a moving average of $\delta_{i,j}^t$ for all values of t, such that $\delta_{i,j}^t \neq \infty$. The reason to average the data is to smooth the effects of single noised readings and for the readings to become closer to

[1]Full paper is available at: http://goo.gl/MynvjM

the mean of the current error. At the same time, the use of moving average vs aggregated average is to avoid historically distant observations to affect the estimations of the current noise levels, which are expected to differ in their mean. The choice of the window size is discussed in the experiments section (section VI). For the theoretical analysis, we will assume the moving average window size was calibrated per camera and is just enough for a confident estimation.

In the example presented in the definitions section (Fig. 1), assume r_4 observed r_2 at $(-0.9, 2.25)$ and r_3 at $(1, 1.5)$ (at $t = 1$). Then (note that both visible and deviation matrices were reduced to submatrices of the visible robots r_2, and r_3):

$$\widehat{O}_4^1 = < (\infty,\infty), (-.9, 2.25), (1, 1.5), (0,0) >,$$

$$V_4^1 = \begin{bmatrix} (0,0) & (1.9,-.75) \\ (-1.9,.75) & (0,0) \end{bmatrix}$$

and

$$\Delta_4^1 = \begin{bmatrix} 0 & \sqrt{.1^2+.75^2} \\ \sqrt{.1^2+.75^2} & 0 \end{bmatrix} \approx \begin{bmatrix} 0 & .757 \\ .757 & 0 \end{bmatrix}$$

To summarize, each robot r_i estimates the positions of the robots that are within its sensing range and derives a moving average relative error matrix AD , while $\overline{\Delta}_t$ captures the true relative positional errors of all robots in the formation at times $t \in \{1, 2, \dots\}$ and is used as the distance matrix argument to the `TreeReconstruction` algorithm.

V. ULS AND UVLS ALGORITHMS

In this section we present the `ULS` and `UVLS` algorithms, that utilize the `TreeReconstruction` algorithm described in section IV. Both `ULS` and `UVLS` algorithms are very similar in their preprocessing of data: They begin by using the *AD* matrix as the input distance matrix to the `TreeReconstruction` algorithm. Then they use the reconstructed tree to infer about the accumulated error of each vertex to the assumed global leader (GL) by calculating their path lengths from it. The following `CoreLS` function is capturing the above, returning a mapping from visible robots ids to path lengths:

Algorithm 3 CoreLS(accepts U, assumes access to AD)

```
tree, root ← RECONSTRUCTTREE(U, AD)
M ← TREE.GET_DISTANCE_MATRIX()
W ← ∅
for i from 1 to U.size() do
    W[U[i]] ← M[root.id][i]
end for
return W
```

Note that although `TreeReconstruction` reconstructs an undirected tree, the conversion to a directed one is natural, from GL (root) to the visible robots (leaves).

A. *ULS Algortihm*

The `ULS` algorithm is executing `CoreLS`, and selects the visible robot with minimal vertex weight, expecting it to be the one with minimal accumulated error:

Algorithm 4 ULS(assumes access to the visible set U)

```
W ← CORELS(U)
robot ← U[0]
for i from 1 to U.size() do
    if W[U[i]] < W[robot] then
        robot ← U[i]
    end if
end for
return robot.id
```

B. *UVLS Algorithm*

The `UVLS` algorithm calculates the expected pose of the robot from each visible robot (using F_{orig}) and performs a weighted average of the resulted positions, using the inverses of the weights obtained from the execution of `CoreLS` (so the shorter the path from a visible robot to the GL, the smaller its error and the higher its impact on the average):

Algorithm 5 UVLS(assumes access to U and to F_{orig} as F)

```
W ← CORELS(U)
pose ← 0⃗
sum ← 0
for i from 0 to U.size() do
    w ← (W[U[i]])^{-1}
    pose ← w * (pose + F[i][ID])    ▷ ID of the executor.
    sum ← sum + w
end for
return pose/sum
```

C. *Correctness and accuracy analysis*

Lemma 5.1: The positional error of any robot r_i from the global leader is the sum of the positional errors of all robots in the control path of r_i.

Proof: Proof by induction over l - the length of a path between r_i and the GL in the control graph: For $l = 1$, r_i's local leader is the global leader r_g and its pose estimation is $p_g + Err_{i,g}$ by definition (actual position of r_g and the estimation error of that position made by r_i). We assume for $l \in \mathbb{N}$ that the control path of r_i is $\{r_i = r_{j_0}, \dots, r_{j_l} = r_g\}$ and that $p_i = p_g + \sum_{t=1}^{l} Err_{j_{t-1}, j_t}$ and prove for $l+1$: Let r_k be the local leader of r_i, then the control path of r_k is coincides with the control path of r_i (but starts from the second element) and is of length l and so $p_k = p_g + \sum_{t=1}^{l+1} Err_{j_{t-1}, j_t}$ by the induction assumption. By definition, $p_i = p_k + Err_{i,k}$ and from the above formula for p_k we get that $p_i = p_g + \sum_{t=0}^{l+1} Err_{j_{t-1}, j_t}$ as required. ■

Lemma 5.2: The average $\delta_{i,j}$ over time is proportional to l - the length of the path between r_i and r_j in the control graph, i.e. it approaches $E[S_{Err}] * l$, where S_{Err} is the minimal noise mean possible for the robot.

Proof: by induction on l: For $l = 1$, either r_i is the local leader of r_j or vice versa. Assume *w.l.o.g.* that r_j is the local leader of r_i, then $\delta_{i,j} = S_{Err}$. The average S_{Err} approaches the expected value as time increases, i.e. $\frac{1}{t}\sum S_{Err} \xrightarrow[t\to\infty]{} E[S_{Err}] = E[S_{Err}] * 1 = E[S_{Err}] * l$ and so the claim is true.

Assume the statement is correct for any r_i and r_j with path length of size $l > 0$ and proof for $l+1$: Since $l > 0$, $l + 1 > 1$

which means that *w.l.o.g.* r_i has a local leader r_a and $a \neq j$. By the induction assumption, $\lim_{t\to\infty} \frac{1}{t}\sum \delta_{a,j} = E[S_{Err}] * l$ and since r_a is the local leader of r_i, $\delta_{i,a} \to E[S_{Err}]$, it follows that $\frac{1}{t}\sum \delta_{i,j} = \frac{1}{t}[\sum \delta_{i,a} + \sum \delta_{a,j}] \to E[S_{Err}] * (l+1)$. ■

Theorem 5.3: `CoreLS` correctly identifies a subtree of the control graph whose leaves are contained in the visible set.

Proof: By lemma 5.2, given enough observations for the moving average to become close enough to the expected value, it is possible to obtain the distance matrix of the visible robots in the control graph. From section IV-A follows that `TreeReconstruction` will correctly reconstruct T - the subtree of the control graph, such that $L(T) \subset U$. ■

The following Corollary summarizes the theoretically proven deviation convergence of `ULS` and `UVLS` :

Corollary 3.1: Assuming the root was correctly identified, we can conclude from lemma 5.1 that both algorithms obtain the correct estimation regarding the magnitude of the accumulative error of each visible robot from 3, and hence `ULS` selects the visible robot with smallest error and `UVLS` provides a pose derived from the visible set, such that for every visible robot, its weight is in inverse ratio to its error.

D. Complexity analysis

Given k to be the size of the visible set, `CoreLS` algorithm executes `TreeReconstruction` in $O(k^3)$ (as explained in IV-B) and traverses over each of the vertices exactly once, adding $O(k)$ (since the number of vertices in the tree returned by `TreeReconstruction` is at most $2k$). Both `ULS` and `UVLS` execute `CoreLS` and traverse over each visible robot once, adding another $O(k)$. Therefore that the runtime of both `ULS` and `UVLS` is bounded by the runtime complexity of `TreeReconstruction` i.e. by $O(k^3)$.

The complexity of the space used by `TreeReconstruction` is $O(k^2)$. The rest of the data structures used are the weight list in `CoreLS`, that requires $O(k)$ space, while `ULS` and `UVLS` are adding $O(1)$ space on top of that. We conclude that the space complexity of both `ULS` and `UVLS` is $O(k^2)$.

VI. Empirical Evaluation

The experiments to support the theoretical findings in this paper were performed using the ROS/Gazebo simulator[2][3], simulating the behavior of Hamster robots[4].

The sensing error of each robot was modeled by a random variable with normal distribution, with mean dependent on the distance and angle to the observed robots. Additionally, the mean of the noise changed over time to simulate dynamics of changes in the sensing conditions, which affect the estimation errors of the robots.

Our experiments compared moving average window size to derive *AD* with values of 10, 50 and 100, which provided standard error of the true mean estimations of $\frac{s}{3}$, $\frac{s}{7}$, and $\frac{s}{10}$ respectively, where s is the sample deviation.

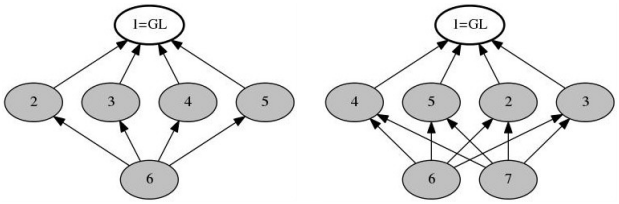
We used two types of navigation patterns: In the first case, the GL was driving forward, and in the second it would turn every 10 seconds to interchanging directions, simulating navigation in a world with obstacles.

The experiments were conducted with team sizes of 6 and 7 that were given a formation to maintain that assured GL is a non-trivial vertex in the control graphs.

All experiments were using the same PD controller to advance to a desired location, which was either obtained directly (from `UVLS`), or derived from the local leader.

The experiments compared `ULS` and `UVLS` to a centralized approach, where there is perfect communication between the GL and the other robots. GL checked the noise level (mean) between every pair of robots (as was reported by the noise simulation unit) and assigned the local leaders to all robots in the team. It is important to note that the centralized approach provided an optimal local leader assignment, which does not guarantee the most accurate formation in noisy environments.

One of the formations can be seen in Figure3. Each method (Centralized, `ULS` , and `UVLS`) was executed 100 times for each combination of the parameters. In all experiments, robot #5 did not have an over time increasing noise model and the noise of its sensor readings was dependent on distance and angle to the observed robot only. The centralized approach selected it as the best local leader for robot #6 (and for robot #7 in the 7-robots experiment), and the expectation was for `ULS` to select the same robot.



(a) 6 robots formation (b) 7 robots formation

Fig. 3. An example of the formation patterns and the visibility graphs used in the experiments. In this case, robot 1 is the GL and the local leader of robots $2-5$. Robot 6 in 3(a) (and 7 in 3(b)) can choose any of robots $2-5$ to follow (or derive a virtual leader).

The true distances between each robot and the global leader were recorded at 5Hz rate during the execution and the overall formation error was calculated as the norm of the vector of deviations of each robot from its expected distance to the GL. Similar experiments results were averaged over the time axis to compare between the performance of every method that was put into test.

In all scenarios with linear forward movement, `ULS` performed similarly to the centralized algorithm (indistinguishable by ANOVA test), while `UVLS` was statistically significantly better (ANOVA test, p-value < 0.001). In the obstacle-avoiding navigation scenario, all three (`ULS` , `UVLS` and centralized) were indistinguishable in their average performance (p-value > 0.5) in the 7-robot formation case, and in the 6-robot formation the centralized method performed significantly better than the `UVLS` , that itself performed statistically significantly better than the `ULS` algorithm (both with p-value < 0.001).

Below are the results for 6 and 7 robots formations.All of them used moving average window size of 10. The other window sizes that were tested, provided similar results, with

[2]http://ros.wiki.com

[3]http://gazebosim.org

[4]http://www.cogniteam.com/hamster4.html

higher variance for each method.

During the first 5 seconds of execution of every experiment, the robots were staying still. This was done to overcome issues with late loading of some of the simulated modules, due to the large number of components, simulated by the Gazebo simulator. Typically, after 3-4 seconds the experiment was up and running, however the movement was not smooth, until

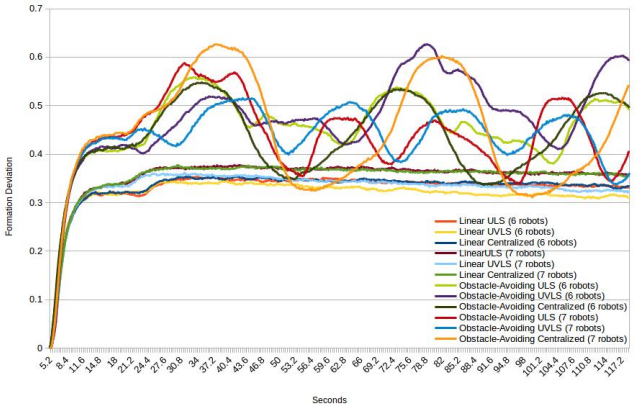


Fig. 4. Average formation deviations over time (lower values correlate to more accurate formation control, i.e., better performance)

Another problem experienced during the runs was occasional disconnections of the last robot from the rest of the formation during the first turn. There is an ongoing work to optimize the implementation of the leader-following mechanism to overcome rough movements. We report herein the results of all successful experiments (that did not result in such disconnected formations).

To conclude, `ULS` provided excellent results while the formation moved forward only, but had multiple problems during turns and might not be as suitable for high-rate dynamically changing environments. There is currently an ongoing work that aims to improve `ULS` to make it more prone to disconnections during turns. `UVLS` provided even better results, and in fact outperformed the centralized algorithm, due to the fact that the errors of the robots are not correlated. During turns it was inferior to the centralized approach, but provides a better alternative to the `ULS` when communication is absent.

VII. Conclusions and Future Work

In this paper we have presented two approaches for solving the formation control problem distributively, using no explicit communication between the team members: uncertain leader selection (using the `ULS` algorithm), and uncertain virtual leader selection (using the `UVLS` algorithm). Both solutions are based on a tree-reconstruction algorithm, that recovers the tree representing the hierarchy of local leaders in the formation. We have shown that the algorithms converge to a minimal deviation from the desired formation.

For future work, we plan to extend this work by providing empirical results performed on real robots, while modeling non-holonomic estimation errors (i.e. S^i_{Err} for every robot r_i). Additionally, since the main contributor to the runtime complexity of both algorithms is `TreeReconstruction` there is an ongoing work to replace it with a more efficient algorithm, that would not not based on `NeighborJoining`

References

[1] Javier Alonso-Mora, Eduardo Montijano, Mac Schwager, and Daniela Rus. Distributed multi-robot formation control among obstacles: A geometric and optimization approach with consensus. In *In proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5356–5363, 2016.

[2] Tucker Balch and Ronald C Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[3] Alyxander Burns, Bernd Schulze, and Audrey St John. Persistent multi-robot formations with redundancy. *Proceedings of 13th International Symposium on Distributed Autonomous Robotic Systems*, 2016.

[4] Aveek K Das, Rafael Fierro, Vijay Kumar, James P Ostrowski, John Spletzer, and Camillo J Taylor. A vision-based formation control framework. *IEEE transactions on robotics and automation*, 18(5):813–825, 2002.

[5] Vali Derhami and Yusef Momeni. Applying reinforcement learning in formation control of agents. In *Intelligent Distributed Computing IX*, pages 297–307. 2016.

[6] Jaydev P Desai, James P Ostrowski, and Vijay Kumar. Modeling and control of formations of nonholonomic mobile robots. *IEEE transactions on Robotics and Automation*, 17(6):905–908, 2001.

[7] Jakob Fredslund and Maja J Mataric. A general algorithm for robot formations using local sensing and minimal communication. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, 2002.

[8] Nicolas Gallardo, Karthik Pai, Berat A Erol, Patrick Benavidez, and Mo Jamshidi. Formation control implementation using kobuki turtlebots and parrot bebop drone. In *Proceedings of the World Automation Congress (WAC)*, pages 1–6, 2016.

[9] Jing Guo, Zhiyun Lin, Ming Cao, and Gangfeng Yan. Adaptive leader-follower formation control for autonomous mobile robots. In *Proceedings of the American Control Conference (ACC)*, pages 6822–6827, 2010.

[10] Gal A Kaminka, Ilan Lupu, and Noa Agmon. Construction of optimal control graphs in multi-robot systems. *Proceedings of 13th International Symposium on Distributed Autonomous Robotic Systems*, 2016.

[11] Gal A Kaminka, Ruti Schechter-Glick, and Vladimir Sadov. Using sensor morphology for multirobot formations. *IEEE Transactions on Robotics*, 24(2):271–282, 2008.

[12] M Anthony Lewis and Kar-Han Tan. High precision formation control of mobile robots using virtual structures. *Autonomous robots*, 4(4):387–403, 1997.

[13] Claudio Paliotta and Kristin Y Pettersen. Leader-follower synchronization with disturbance rejection. In *Proceedings of the IEEE Conference on Control Applications (CCA)*, pages 360–367, 2016.

[14] Long Peng, Fei Guan, Luc Perneel, Hasan Fayyad-Kazan, and Martin Timmerman. Decentralized multi-robot formation control with communication delay and asynchronous clock. *Journal of Intelligent & Robotic Systems*, pages 1–20, 2017.

[15] Naruya Saitou and Masatoshi Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.

[16] Frank E Schneider and Dennis Wildermuth. A potential field based approach to multi robot formation navigation. In *Proceedings of the IEEE International Conference on Robotics, Intelligent Systems and Signal Processing*, volume 1, pages 680–685, 2003.

[17] James A Studier, Karl J Keppler, et al. A note on the neighbor-joining algorithm of saitou and nei. *Molecular biology and evolution*, 5(6):729–731, 1988.

[18] José Vilca, Lounis Adouane, and Youcef Mezouar. Adaptive leader-follower formation in cluttered environment using dynamic target reconfiguration. In *Distributed Autonomous Robotic Systems*, pages 237–254. 2016.

[19] Zhihao Xu, Martin Schröter, Dan Necsulescu, Lei Ma, and Klaus Schilling. Formation control of car-like autonomous vehicles under communication delay. In *Proceedings of the IEEE 31st Chinese Control Conference (CCC)*, pages 6376–6383, 2012.

[20] Guoyu Zuo, Jiatong Han, and Guansheng Han. Multi-robot formation control using reinforcement learning method. In *Proceedings of the International Conference in Swarm Intelligence*, pages 667–674, 2010.