Shane Benetz
13 June, 2023
MAE 247

Final Report - Formation Control Algorithm

Introduction:

Multi-agent systems involve the coordination and collaboration of multiple intelligent agents to accomplish complex tasks efficiently. One compelling area of research within this domain is multi-agent formation control, which focuses on achieving coordinated movements and formations among a group of autonomous agents. Formation control refers to the ability of a group of agents to organize themselves into specific arrangements or patterns while maintaining a desired spatial relationship between each other or with regards to an environment. By achieving precise formation control, these systems can enhance their capabilities, such as collaborative exploration, cooperative surveillance, efficient transportation, and targeted communication. The goal of this report is to discuss how the paper *Global and Robust Formation-Shape Stabilization of Relative Sensing Networks* by Jorge Cortez attempts to solve this problem of formation coordination using a distributed algorithm in a system of agents that are connected through their ability to sense each other or communicate between agents. The proposed strategy claims to offer a discrete-time distributed algorithm that stabilizes a group of agents to a desired formation no matter the initial configuration, granted the graph topology between agents has a globally reachable node. It also claims to be robust to sensing or connection failures and measurement errors with regards to convergence.

State of the Art:

There have been a handful of methods proposed over the years that solve the problem of formation control. Some of the ones discussed in the literature review for this paper are continuous-time formation control using algebraic graph-theoretic tools, hybrid rendezvous-to-formation control, and shape stabilization through graph rigidity. More recently, developments in the area of reinforcement learning and deep learning have created space for many more approaches to arise.

The first of these methods leverages very well known graph theory principles paired with dynamic control in continuous time. Since the state of our system can be represented by a vector/matrix, we can use popular matrix dynamic control to control the state over time. Each agent is a part of the system state, and each agent's state can be multi-dimensions itself. So using matrix dynamic control, the goal is to find a controller , whether first order, second order, or beyond, that will cause motion that ends in the desired formation. These types of algorithms pair dynamic system control in continuous time with basic graph theory where G is a graph with nodes V and edges Ɛ (G={V,Ɛ}). Using this graph's Laplacian matrix and a tunable controller, we

can create a continuous function for the state that begins at an initial state and converges to a desired end state. An example format of this problem is as described below.
Each agent has the same dynamics given by:

$$\dot{x}_i = A_{veh}x_i + B_{veh}u_i \qquad i = 1 \dots N \quad x_i \in \mathbb{R}^{2n}$$

We can model the output function as related to the distance of the current state to the desired state h:

$$y_i = (x_i - h_i) - \frac{1}{|\mathbb{J}_i|}\sum_{j\in\mathbb{J}_i}(x_j - h_j) \qquad i = 1, \dots, N.$$

If we compile that for the whole system, we get a system like this:

$$\dot{x} = Ax + BFL(x - h)$$

where F is the controller we must tune for our problem[1]. Although this type of solution is successful, it has some limitations. The first drawback is that each agent needs the same exact linear dynamics, so we cannot be that robust in what kind of systems we control. The other major drawback is that we must design a controller for each problem that will be stable and cause convergence for the given system.

Another algorithm described in the literature review section is called hybrid rendezvous-to-formation control.[2] This strategy looks to solve edge loss and connectivity of non-static graph topologies problems. The first main point of the algorithm is that the agents must rendezvous at a location in order to form a complete connectivity graph. Once they have rendezvoused, they can then form a spanning tree of edges that works for desired formation. Using this, it becomes a dynamic control problem again. They use an "edge-tension" function which basically keeps distance between agents below a certain threshold to keep the motion correct. This function also is used to produce a control law for them, still using continuous dynamic control. Even though each agent must have the same dynamics, this can be done in a distributed way. It also solves the initialization problem since once the rendezvous happens, the connectivity of the graph can be much more easily controlled. Although this algorithm has robust dynamics that can account for edge losses and non-static connectivity, the difficulty and extra work comes in the rendezvous of the agents which is a different sector of strategies in itself. It also requires extra time to run this part, requires a synchronized switch from rendezvousing to formation control, and still holds the problems of continuous time feedback controllers. These are some of the problems that we will also have to deal with in the paper of interest.

The final type of strategy in the literature review is one that deals with stabilization through graph rigidity. A formation is considered rigid if the inter-agent distances and angles are

[1] Lafferriere, Gerardo; Williams, Anca; Caughman, John S. IV; and Veerman, J. J. P., "Decentralized Control of Vehicle Formations" (2004). Mathematics and Statistics Faculty Publications and Presentations. 142.
[2] M. Ji and M. Egerstedt. Distributed control of multiagent systems while preserving connectedness. IEEE Transactions on Robotics, 23(4):693–703, 2007

fixed and cannot be changed without introducing deformation. A rigid formation allows for precise control over the relative positions of the agents, enabling them to achieve coordinated movements. For these types of problems, a potential function or a Lyapunov function is used to measure the system's current state. If these functions are decreasing over time, that means that the formation is converging to the correct inter-agent distances and therefore relative positions. A common way to solve the problem is through using Jacobian gradient descent for the control law. Although this can be successful, one of the main problems that is run into is undesired local minima. Using the gradient descent, it is possible that the system will get stuck at a minimum that is not actually the desired one. This is what the paper of interest will try to solve in its strategy to create a consistent desired convergence.

The final category of solutions to the formation control problem is that of the more recent reinforcement learning and deep learning tactics. One strategy used is Policy Learning where RL algorithms can be utilized to learn optimal policies for agent behavior, learning through trial and error and receiving rewards. An additional strategy is Adaptive Control where learning algorithms have agents that can learn models of their own dynamics and the dynamics of other agents, allowing for control strategies to adjust in real-time. One more tactic is Trajectory Prediction where learning techniques can be applied to predict the future trajectories of agents in a formation. By analyzing historical data and training predictive models, agents can anticipate the movements of other agents and adjust their control actions accordingly. All of these require more time and effort to train on data, but they do offer a more adaptive approach to the problem. These algorithms aren't as able to be validated through simple mathematical proofs, but rather employ a strategy to incrementally improve over time.

Main Algorithm:

This proposed algorithm approaches the formation control problem in a few specific ways: it is a discrete-time algorithm, it works in multiple dimensions, it only requires one globally reachable node in the connectivity graph, and it uses Jacobi over-relaxation to minimize a stress-majorization function which gives the solution to the control problem. The benefit of a discrete-time algorithm is that it is usually less computationally expensive and can actually be computed on a robot with a specific internal clock. This also pairs well with Jacobi Over-relaxation which runs incrementally. Using this on the stress-majorization function allows us to not get stuck at local minima while trying to converge to the actual solution (more details on this in analysis). Because of that, using these two things saves us from needing the approach of a continuous time controller which is more difficult to design and may not converge to the actual desired formation. On top of that, the value of being a multidimensional algorithm is that we can apply it to both simple and complex problems where the state must be described by more than just one or two variables. Also, even though we still have a requirement of a graph that has a globally reachable node, this is one of the looser constraints for graphing algorithms in this field of study.

Distributed algorithms are useful in multi-agent control because centralized computation is much more difficult for complex cases. In the non-theoretical world, there are physical barriers to keep in consideration, such as how far away an agent can actually sense its neighbors or how far apart every agent can get while still being in contact. In centralized algorithms, we must compile all data into one place and run computation, which is even harder with these communication difficulties. So, running distributed algorithms like these helps systems run more efficiently while trying to pass around less data overall and having each agent do its own computation. This becomes even more beneficial the more complex an algorithm is, because we are able to split up the computation responsibility over a whole system, which is one of the biggest reasons to use multi-agent systems in the first place, besides just for the purpose of information gathering.

To demonstrate this algorithm we will use a 2D example that will show the main steps of what the algorithm entails. The main algorithm can be mostly separated into two parts: frame alignment and Jacobi control. The first requirement for the control algorithm is that each agent must have the same reference frame. Since each agent can start in any location and in any direction, we must apply an algorithm that aligns them. We basically use a wave of repetitive motions that get performed starting with the globally reachable node. That node moves along its axes repetitively while its neighbors sense those directional movements and align their axes with the sensed ones. The newly aligned agents then also perform the motion until all robots share the same reference frame.

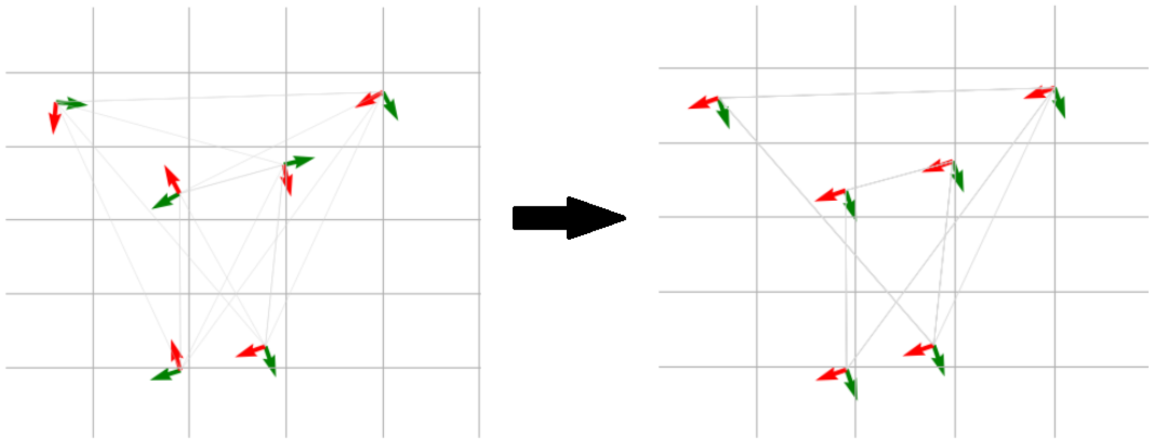


Figure 1: Agents start with random orientation and position, then become aligned in their axes

Once the agents have aligned their frames, then they are ready to start moving toward the desired formation. Using an update equation, each agent then computes its own control by considering their own position, their neighbors' sensed positions, and the distance between the current locations and the desired formation. Over time, the system moves toward the goal formation, and even though the formation is specified by points in space, it is not guaranteed that the system will end in that location, but we are guaranteed that they will converge to some rotated or translated version of the formation.

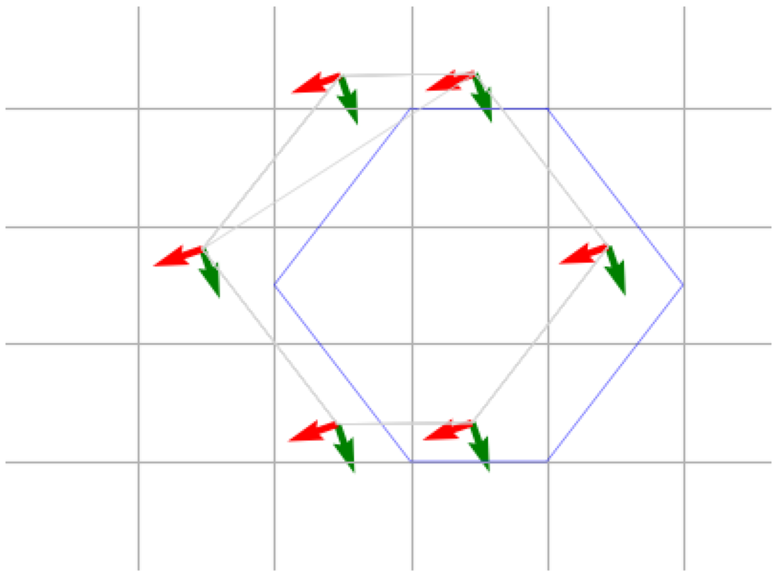


Figure 2: The agents converge to a formation matching the desired (blue), but they may not be in the same location

Analysis:

Firstly, we will go into a more detailed description of the algorithm in each of its parts. We must add a little background information to actually understand what is going on behind the scenes. We will talk about the concepts of reference frames, graph theory, the problem formation, and stress-majorization to solve the problem.

This algorithm heavily relies on the principles of reference frames to make it actually work in a distributed way. A reference frame is simply a point in space paired with an orientation that can be defined by a set of axes vectors. A fixed frame is connected to a point in space and does not change (hence the name). A body frame is connected to an entity and moves as that entity moves. Knowing that this adds complexity, the algorithm is much easier to implement in a centralized way in simulation since we decide a reference frame while making the graph we represent it on. If we consider agents only able to sense neighbors in real life and have no other real global information, then the ability to have this fixed reference frame does not exist. So, everything must be computed in the body frame. We are able to represent the difference between two orientations of frames by a rotation matrix, which transforms the vectors of one orientation to the other frame's orientation vectors.

The next thing this relies on, like most other multi-agent algorithms, is graph theory. We can represent the system by a directed graph (G) where each agent is a node (v_i) in the set of vertices (V), and when one agent(v_i) can sense or communicate with another node(v_j), the graph has an edge (i,j) in the set of edges ($\mathcal{E}$). The out-degree of a node (v_i) is how many edges in $\mathcal{E}$ connect from v_i to another node. The in-degree of a node (v_i) is the inverse, where it is the number of edges that connect from any other node to node v_i. When the out-degree equals the in-degree we have an undirected graph. We can also define an adjacency matrix A(G) where $a_{ij}>0$ if edge (i,j) exists. Using these properties, we can get the Laplacian matrix L(G) which is

$D_{out}(G)$ - A(G) where $D_{out}(G)$ is a diagonalization of the vector of out-degrees for each node. If this matrix L(G) has rank = n-1, that means that there exists a globally reachable node in the graph, which in other terms means that starting at any other node, we can traverse edges in Ɛ and end up in that globally reachable node. Other important graph theories for this algorithm are the mirror and reverse graphs. A reverse digraph (Ğ) reverses the order of all edges in G (i,j) → (j,i). Some other properties of this are $A(\breve{G}) = A(G)^T$ and $L(\breve{G}) = D_{out}(\breve{G}) - A(\breve{G}) = Din(G) - A(G)^T$. A mirror digraph (Ĝ) takes the union of edges from G and Ğ, essentially converting the graph to an undirected graph. Some properties are $A(\hat{G}) = 0.5(A(G) + A(G)^T)$ and $L(\hat{G}) = 0.5(L(G) + L(\breve{G}))$.

How we set up our problem using these two things is that we have a group of N agents whose states are captured individually by $p_i \in R^d$. Each has its own body reference frame and computes its own control input ($u_i \in R^d$), which is just the change in its local frame. Since the goal is formation control, we represent a vector of goal states as the configuration:

$$Z^* = (z_1^*, \ldots, z_n^*) \in (R^d)^n.$$

We can also define an adjacency matrix K where K_{ij} is the distance in space between nodes v_i and v_j and is only non-zero if an edge exists. Let Rgd(Z*) be the set of all configurations that can be achieved by simply rotating or translating Z*. Each of these configurations has the same inter-agent distance matrix K.

The last major piece to understand is a strategy called “stress majorization.” To use this, we must define a stress function for our problem. The most basic one we can use is given by:

$$\text{Stress}_G(p_1, \ldots, p_n) = \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} (\|p_i - p_j\| - \Bbbk_{ij})^2. \quad (2)$$

Given any states $Z = (z_1, \ldots, z_n) \in (R^d)^n$ and unweighted graph G, G^Z is the weighted version of G with the adjacency $A(G^Z)$:

$$a_{ij} = \begin{cases} \Bbbk_{ij} \operatorname{inv}(\|z_i - z_j\|), & (i,j) \in \mathcal{E}, \\ 0, & (i,j) \notin \mathcal{E}, \end{cases} \quad \operatorname{inv}(x) = 1/x \text{ if } x \neq 0, \text{ and } \operatorname{inv}(0) = 0$$

This new graph basically defines the ratio of current inter-agent distance to the desired inter-agent distance. Therefore, for all Z ∈ Rgd(Z*), $G = G^Z$.

We want a function to perform gradient control on that will not converge to anything else besides a configuration in Rgd(Z*). We then redefine the stress function in a way that still converges to the same values, but has properties that won't converge incorrectly:

$$\mathcal{F}_G^Z(P) = \operatorname{tr}(P^T L(G) P) - 2 \operatorname{tr}(P^T L(G^Z) Z) + \frac{1}{2} \sum_{(i,j)\in\mathcal{E}} \Bbbk_{ij}^2$$

We get this function using some provisions in the paper.
The useful one is Lemma 2.2:
L(Ĝ) is Laplacian of Mirror digraph(Ĝ)

Lemma 2.2 *Given a weighted digraph G and* $x, y \in \mathbb{R}^n$

$$\frac{1}{2} \sum_{(i,j)\in\mathcal{E}} a_{ij}(x_i - x_j)(y_i - y_j) = y^T L(\widehat{G}) x.$$

We get the given stress majorization function above by starting with the original stress function (2) and expanding it to:

$$\text{Stress}_G\,(p_1,\ldots,p_n) = \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} \|p_i - p_j\|^2 - \sum_{(i,j)\in\mathcal{E}} \|p_i - p_j\|\Bbbk_{ij} + \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} \Bbbk_{ij}^2$$

Using Lemma 2.2 we have the equality:

$$\frac{1}{2}\sum_{(i,j)\in\mathcal{E}} \|p_i - p_j\|^2 = \operatorname{tr}\left(P^T L(\widehat{G}) P\right)$$

Expanding the second term of the function we get:

$$-\sum_{(i,j)\in\mathcal{E}} \|p_i - p_j\|\Bbbk_{ij} = -\sum_{|(i,j)\in\mathcal{E}} \Bbbk_{ij}\,\text{inv}(\|z_i - z_j\|)\|p_i - p_j\| \cdot \|z_i - z_j\| < 2\operatorname{tr}\left(P^T L(\widehat{G}^Z) Z\right)$$

This is how we get all the terms of the stress majorization function, and since we have an upper bound in it, we have Proposition 4.1. Proposition 4.2 says that the following holds for the Mirror graph as well

Proposition 4.1 ([4]) *Given an undirected graph G for any $P = (p_1,\ldots,p_n)$, $Z = (z_1,\ldots,z_n) \in (\mathbb{R}^d)^n$,*

$$\textit{Stress}_G\,(p_1,\ldots,p_n) \le \mathcal{F}_G^Z(P).$$

Moreover, if $P = Z$, then $\mathcal{F}_G^P(P) = \textit{Stress}_G\,(P)$.

Proposition 4.2 *Given a digraph G, for any $P = (p_1,\ldots,p_n)$, $Z = (z_1,\ldots,z_n) \in (\mathbb{R}^d)^n$,*

$$\textit{Stress}_G\,(p_1,\ldots,p_n) \le \mathcal{F}_{\widehat{G}}^Z(P).$$

Moreover, if $P = Z$, then $\mathcal{F}_{\widehat{G}}^P(P) = \textit{Stress}_G\,(P)$.

We can write the stress-majorization function out using a Kroneker product instead of a sum:

$$\mathcal{F}^Z(P) = P^T(L(G)\otimes I_d)P - 2P^T(L(G^Z)\otimes I_d)Z + \frac{1}{2}\sum_{(i,j)\in\mathcal{E}} \Bbbk_{ij}^2. \quad (4)$$

Proposition 4.3 shows through calculating the gradient and hessian, this function is convex and therefore won't have unwanted local minima. This is very import and through further calculation, the very important Lemma 4.4 can be shown:

Lemma 4.4 *Given $Z \in \texttt{Rgd}(Z^*)$ and a digraph G with a globally reachable vertex, P is a global minimizer of $\mathcal{F}_{\widehat{G}}^Z$ if and only if*

$$(L(G)\otimes I_d)P = (L(G)\otimes I_d)Z. \quad (6)$$

Now that we have sufficient validation for our distributed control algorithm, we will give a more descriptive explanation of the full algorithm proposed by the paper; first the alignment portion, then the control portion. After that, we will get into the robustness of the algorithm.

The first step is to align the axes of each agent. The importance of this will be demonstrated in the next step. As mentioned earlier, we first start with a globally reachable node. This agent moves some distance in a predetermined direction or set of directions, which corresponds to its axes, then it moves back to its original position. (For example, in 2D, it would move along its X axis, then move back. In 3D, it would move along its X, then return, then along its Y and return again). When one of its neighbors senses that movement pattern, that agent will compute the axes, then rotate itself to match that orientation. Once the orientation is copied, every agent that is aligned will copy the movement patterns. Since we can find the reverse path from a globally reachable node to any other node, we know that this movement and alignment sequence will pass in a wave until all agents are aligned. Although this seems like a time consuming part of the algorithm, the time footprint is really dependent on the graph structure. We could have a system of 100 agents that are in a more connected graph, and their alignment time could be less than that of a graph of 10 agents that is a minimum spanning tree. This algorithm was not a main point of the paper, but just a strategy to set up requirements for the novel algorithm.

Part two of the algorithm is where the real control happens. Using Lemma 4.4, let the following equality hold:

$$b = (b_1, \ldots, b_n) = (L(G) \otimes I_d)Z^*$$

We can then use the following Jacobi iteration algorithm to find the minimum of our stress function given in Lemma 4.4. We can look at it in a few different ways, but two of the important conceptual representations are looking at position changes of agents in a fixed Cartesian coordinate system to understand the correctness, but also in each agent's local frame because that's what is actually being computed in real time. In Cartesian coordinates, there are no frames since everything is basically in a fixed frame. The update for each agent's state is given by:

$$p_i(\ell+1) = (1-h)p_i(\ell) + h\frac{1}{d_i}\Big(\sum_{j\neq i} a_{ij}p_j(\ell) + b_i\Big)$$

If we want to compute the update for each agent in its own frame, the equation is given by:

$$p_i^i(\ell+1) = h\frac{1}{d_i}\Big(\sum_{j\neq i} a_{ij}p_j^i(\ell) + (R_i^{\text{fixed}})^T b_i\Big)$$

Since each of the rotation matrices are the same since the frames are all aligned, R_i^{fixed} is unnecessary and we then have the following update instead:

$$p_i^i(\ell+1) = h\frac{1}{d_i}\Big(\sum_{j\neq i} a_{ij}p_j^i(\ell) + b_i\Big)$$

In this algorithm, *h* is just a scalar value in (0,1) that determines the convergence speed agents to be in the correct positions. Using some manipulations, we can combine the whole system into one update equation to prove convergence. The full system can be written as:

$$P(\ell+1) = P(\ell) - h(D(G)^{-1}L(G) \otimes I_d)(P(\ell) - W)$$

Plugging that back into our stress-majorization function we can get:

$$\begin{aligned}
&\mathcal{F}_G^W(P(\ell+1)) - \mathcal{F}_G^W(P(\ell)) = \\
&\quad = -2hz^T(D(G)^{-1} \otimes I_d)(L(G) \otimes I_d)P(\ell) \\
&\quad\quad + h^2 z^T(D(G)^{-1}L(G)D(G)^{-1} \otimes I_d)z \\
&\quad\quad + 2hz^T(D(G)^{-1} \otimes I_d)(L(G) \otimes I_d)W,
\end{aligned}$$

$$= -hz^T((2D(G)^{-1} - hD(G)^{-1}L(G)D(G)^{-1}) \otimes I_d)z.$$

Using Lemma 2.1 (given in the paper), we can prove that it is a Negative Definite for h in (0,1). That means that the change between time steps over time will slowly converge to zero.
Now we have proved the validity of the algorithm, guaranteeing its convergence under the conditions of a connected graph and the bounds of h. The following is an example of the results provided by the paper:

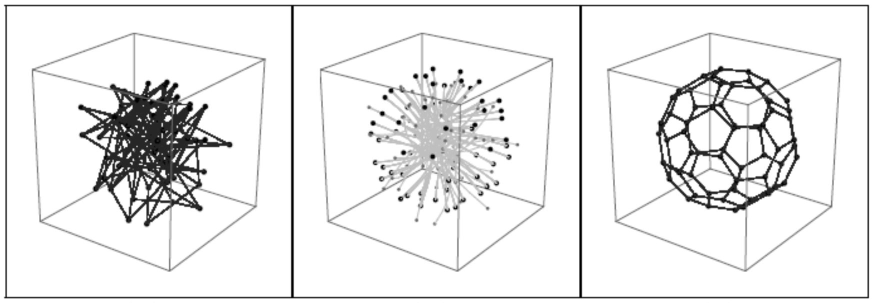
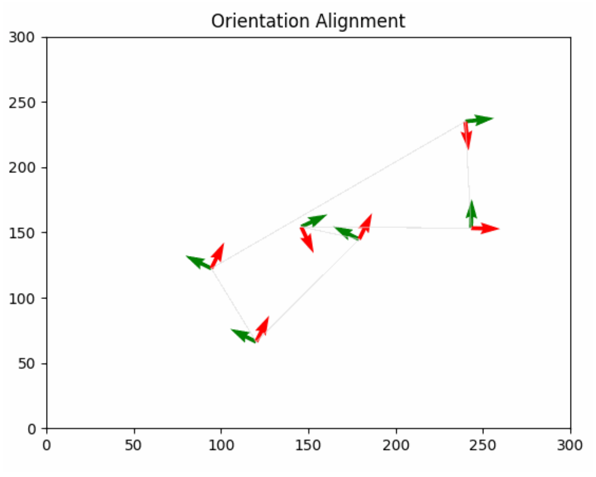


Figure 3: Execution of the algorithm (10) with $h = .25$ over a relative sensing network in $\mathbb{R}^3$ composed of 60 agents. (a) initial configuration, (b) evolution, and (c) final formation.

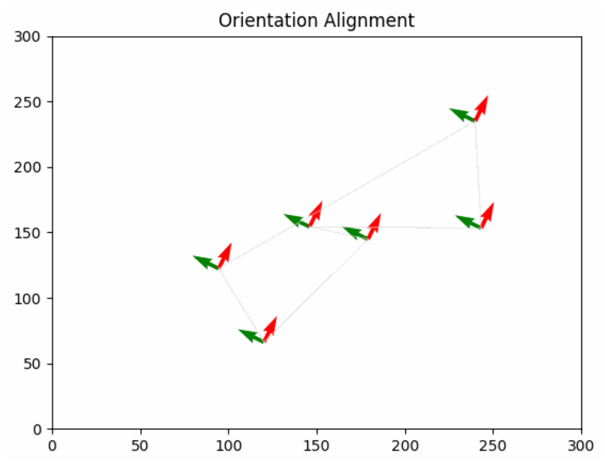
The paper also considered the robustness of this algorithm to errors. The types of errors addressed were link failure, measurement error, and orientation error. Link failure occurs when an agent loses the ability to sense one of its neighbors for some period of time. Basically what happens is that the algorithm runs, but it's treated like each agent is computing its value asynchronously to the others. In this case, agents would be using old values for some neighbor positions. The authors found in their study that anything more than 10 time steps of incorrect positions might lead to failed convergence. Measurement errors describe the imperfect sensing of the robots so that they have a slightly wrong position measurement of their neighbors.
In this case, the algorithm will still converge to a formation with error bounded by the size of how far off their measurements are. The last addressed error robustness was orientation error where agents don't perfectly align themselves in the first step of the algorithm. Even in this case, the algorithm will still converge, but to a slightly deformed formation. Basically it will end up converging to slightly wrong formation, but usually the error is small enough not to be able to tell on a large scale.

Implementation:

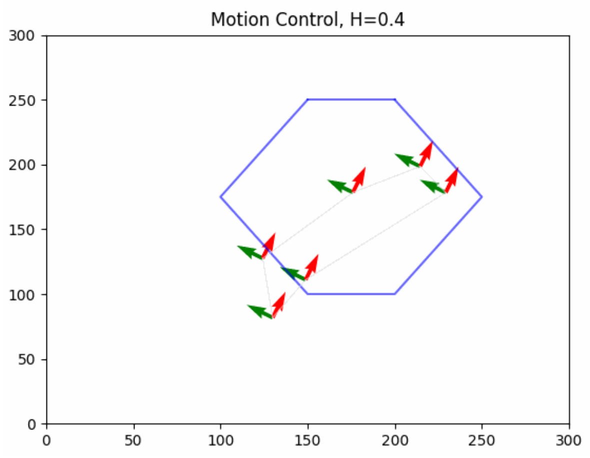
I have implemented this algorithm on a 2D scale simply using Python. My implementation has not been tested on 3D, but in theory it would be fairly easy to extend to that as well. For the first experiment I ran, I did a simple circular graph experiment where each node should have an out-degree of 2. Below are the results:



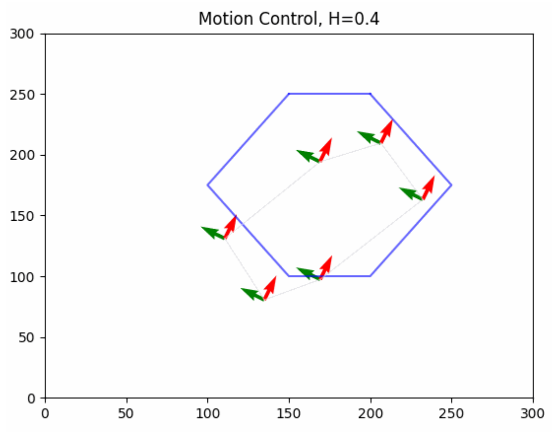
Agent orientation before alignment step



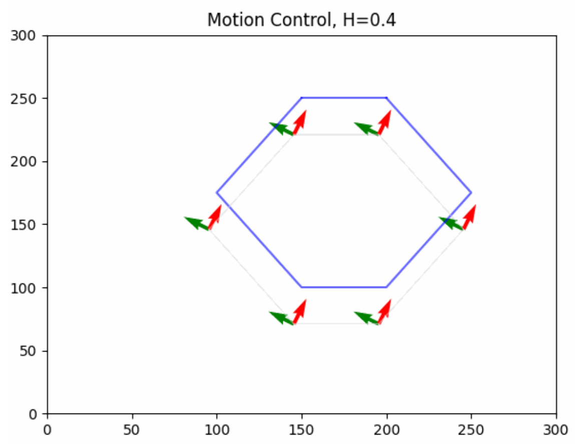
Agent orientation after alignment step



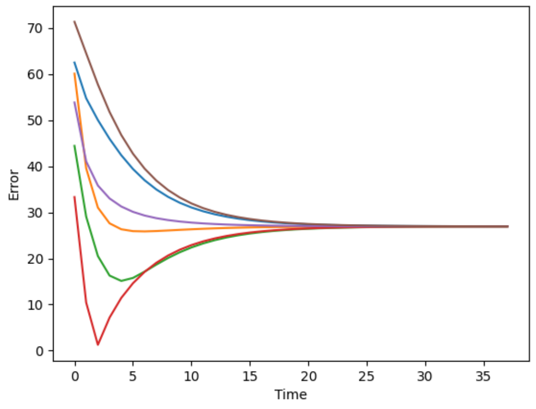
Agent positions before control algorithm



Intermediate step



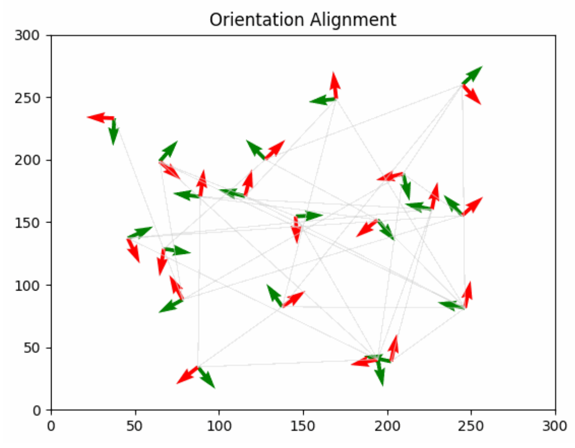
Agent positions after control algorithm



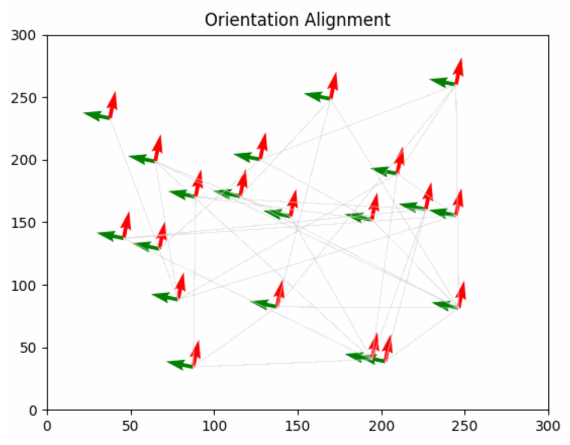
Error from desired formation locations

This experiment converged pretty quickly at around 25 time steps. It led to the correct formation and did not seem to take many wrong directional moves to get there.

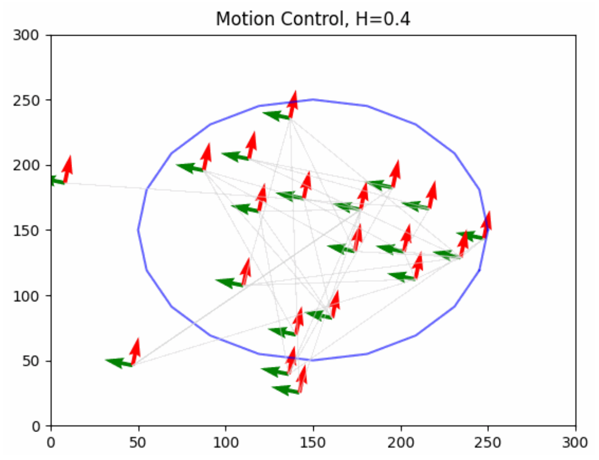
Next, I wanted to experiment with a larger, more dense graph. For this one I created a random undirected graph of size N = 20. It was randomized by randomly adding an edge to another node 15% of the time. The formation shape for this one was a circle pattern. The results are below:



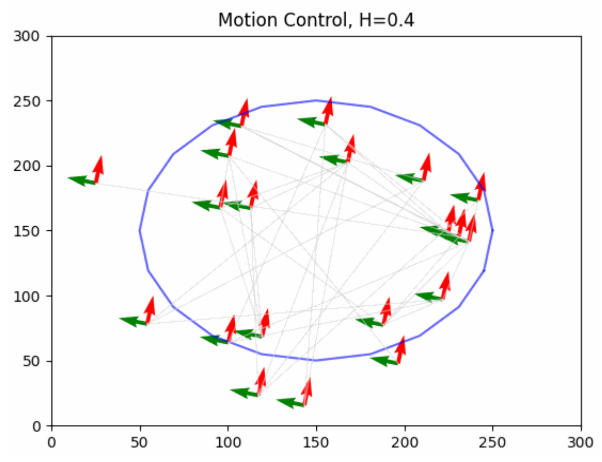
Before aligning agent orientation



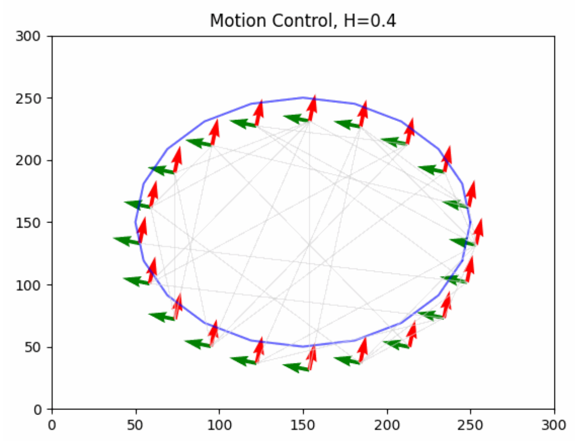
After alignment



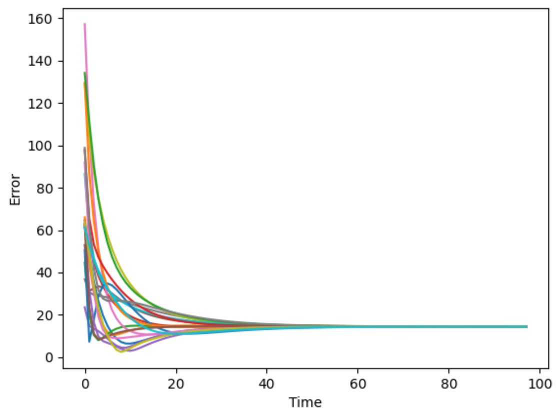
Before control algorithm



Intermediate Step



After control algorithm

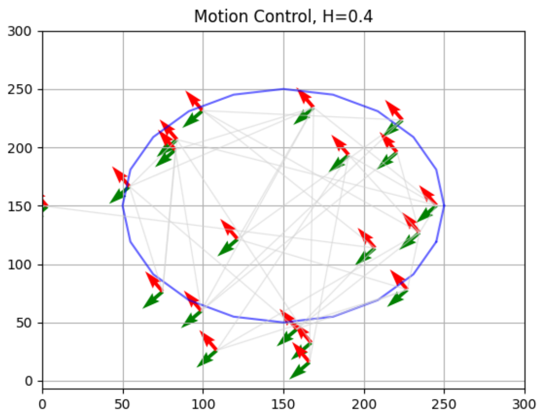


Error from desired locations

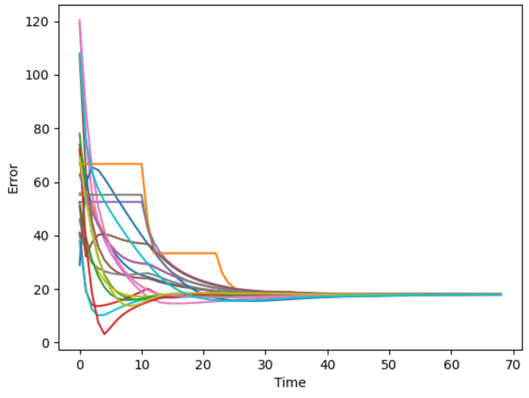
What I learned from this experiment is that even for a larger set of nodes, the algorithm converges fairly quickly still for both alignment and control. This has the same h value for the last experiment and had many more nodes by a factor of around 3.5, and the convergence time was around 2 times as long, so that is a non-linear relationship in a beneficial way.

The final couple experiments I ran with this algorithm had to do with error robustness. I wanted to test what would happen when nodes fully failed, meaning that they just stopped getting new info about all of their neighbors. I also wanted to see what would happen if each node had a chance of losing connection to some of their neighbors for a certain duration of time. For both of these experiments, I had a parameter for failure percentage and duration. Duration decides how long to go before starting to get updated positions of neighboring nodes again.

For the first test called “Delay” I gave a failure percentage of 15%, which was the chance of an agent to fully stop getting new data about nodes for 10 time steps. Some of the results are below:



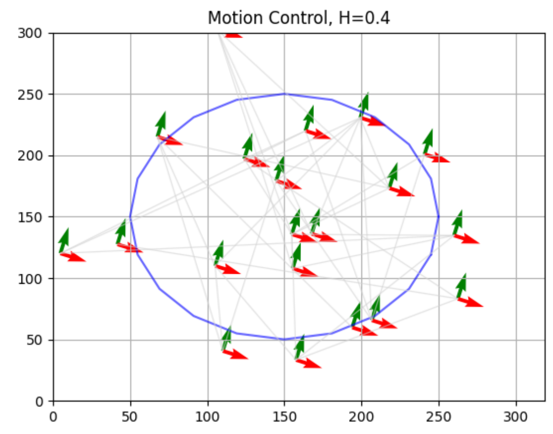
Intermediate time step

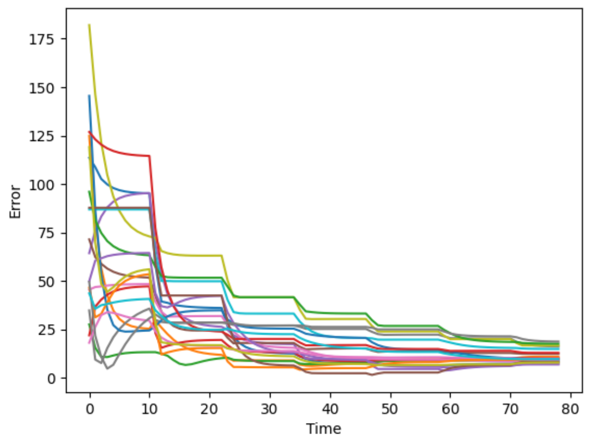


Error from desired formation

What I learned after playing around with parameters was that it always converged and it really didn’t matter the duration of the delay as long as it resumed at some point. The behavior I saw was that the non-delayed agents would move more toward their desired positions, but would slow down and not get all the way there. Then once the failed nodes resumed, the failed nodes would quickly catch up to the rest. You can see in the figure above that there are a few nodes that are more toward the inside or the outside of the forming circle. These are the failed nodes, and basically all they did was cause the system to move toward the right formation, but pause at a deformed state. As seen in the error plot, some of the agents take a stepped approach to the convergence value while others are smooth. This is the difference between the failed and non-failed nodes, since the failed ones basically just freeze where they are.

The next experiment I ran was called “Mixed Delay” where the fail rate was 25%, meaning that for each node there was a 25% chance that a connection it had would fail at any given time step. Once a link had failed, it would remain failed for 10 time steps as well. The results are given below:





Intermediate Step Error from desired formation

What I learned from this experiment was actually quite interesting. What we see in this experiment is also an incremental convergence to the desired formation. The pictured intermediate step exemplifies how some of the nodes could actually get worse (meaning further away from desired) when they had failed links to neighboring nodes. Once the links were restored, they started moving again back toward the goal, but the behavior we see is that in those steps where links have failed, the convergence may not go in the right direction. This property actually affects every agent's convergence instead of just the ones that failed as seen in the step like error plot. The convergence also took much longer for this experiment, and did not fully converge, even though it might have given long enough.

Conclusions:

Overall, I believe that this algorithm proposed by Jorge Cortez is fairly close to as robust as it claims to be. Even though I did not test for really complex graph structures, it can be seen that there aren't big limitations on what kind of desired formation we can give it, since all that is required is rigidity. It also produces a fairly low barrier to entry by only requiring local sensing by agents and a graph structure with one globally reachable node. Even with the claims of robustness, it seems like it is as true in practice as it is in theory. I tried to find parameters in my 2D example that would cause non-convergence, but I really couldn't find any that didn't simply delay the convergence. I'm sure things may have failed more quickly in the 3D case or higher dimensional, but the 2D case at least seemed fairly robust. The claim of convergence no matter the initial condition seems to be pretty true at least in simulation. I am sure in the real world where sensing errors are more pertinent, there leaves room for much more failure. Overall, the algorithm seems to be very scalable as well because of its distributed manner and because the computation is actually not too heavy. That means that it could be applicable in many different scenarios.

There were some shortcomings that I found that were not mentioned in the paper that may have been useful to address. The first that caught my attention is the question of how do the agents know when orientation alignment is done and when it's time to switch to formation control? Another consideration that probably has more of an effect in real world scenarios is the error propagation over time. The longer a system takes to converge, the more likely measurement errors may occur, and those could scale fairly quickly. Additionally, even though edge failures were considered, this really didn't talk about the structure of the graph changing over time. It left me wondering whether convergence still worked if the agents swapped neighbors along the way.

I also have some things that I would be curious to look into more. One of those would be to study the convergence rates as it relates to both graph size and graph connectivity (measured in number of edges). The h value also was a direct control of this, but it would be interesting to see the relationship between h and the actual convergence time. Another thing that piqued my interest was trying to predict where the formation would go based on the initial configuration, since only in the really basic case did it converge to the actual desired locations. On top of that, I'd be curious to know how convergent it would be in the presence of obstacles in the environment since this is a big factor in real world robotics applications. My best guess is that it would still converge to something, but not always the correct formation. All that being said, it was an exciting algorithm to implement and seeing the possibilities of its use interested me to want to experiment with it even more past what has already been done to try to test it in more real applications.

Link to Github with code and result gifs:
https://github.com/sbenetz/MAE247FinalProject

References:

J. Cortes, "Global formation-shape stabilization of relative sensing networks," 2009 American Control Conference, St. Louis, MO, USA, 2009, pp. 1460-1465, doi: 10.1109/ACC.2009.5160240.

Lafferriere, Gerardo; Williams, Anca; Caughman, John S. IV; and Veerman, J. J. P., "Decentralized Control of Vehicle Formations" (2004). Mathematics and Statistics Faculty Publications and Presentations. 142.

M. Ji and M. Egerstedt. Distributed control of multiagent systems while preserving connectedness. IEEE Transactions on Robotics, 23(4):693–703, 2007