CS 455 - Project 1 - MSN Flocking Formation Control - Joseph Trierweiler

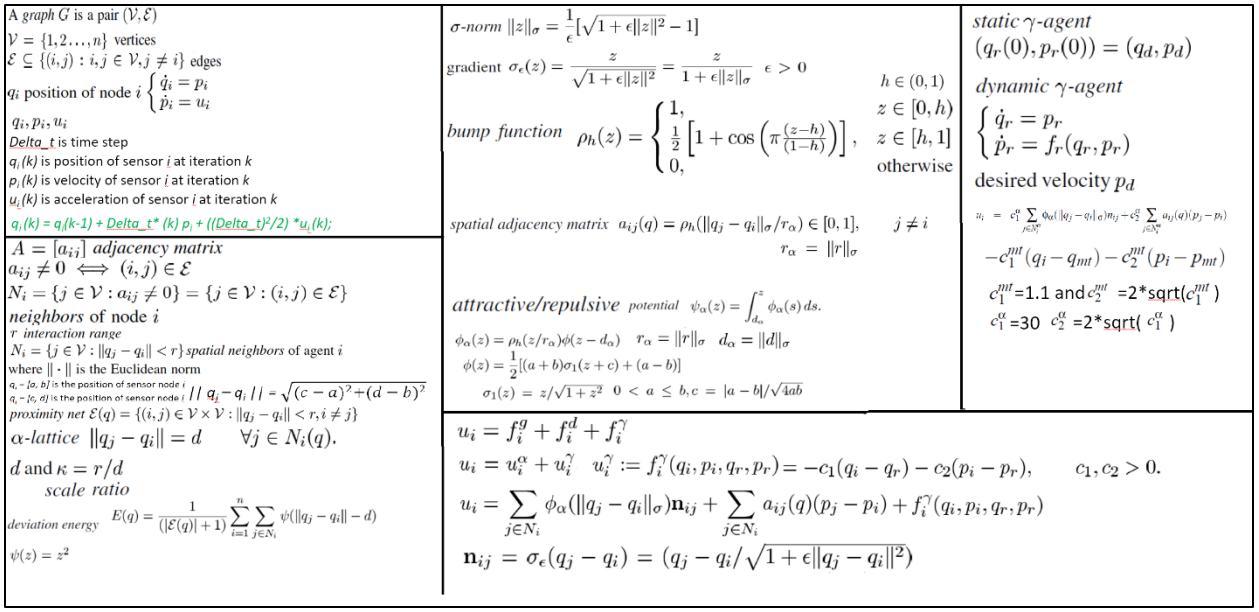
Intro: The purpose of this project is to simulate a Mobile Sensor Network (MSN) that follows a flocking network formation. This flocking network should be in the shape of a quasi-lattice. This shape mimics the shape of flocking networks seen in nature, such as in fish and in ants and other flocking animals. To create these shapes, Reynold's three heuristics must be followed: Flock Centering, Collision Avoidance, and Velocity Matching. All three heuristics are demonstrated in Algorithm 1 and Algorithm 2 found in this project. To test these algorithms for yourself, one needs a copy of MATLAB, and to run main.m with the MATLAB software. Each test case can be tested by modifying the "case_number" variable to match the desired case number. There are several other parameters that can affect the algorithm's behavior that can also be modified that I've listed below.

The parameters used:

- 100 Nodes
- Desired Distance = 15
- C_a1 = 30
- C_mt1 = 1.1
- Epsilon = 0.1
- H = 0.2

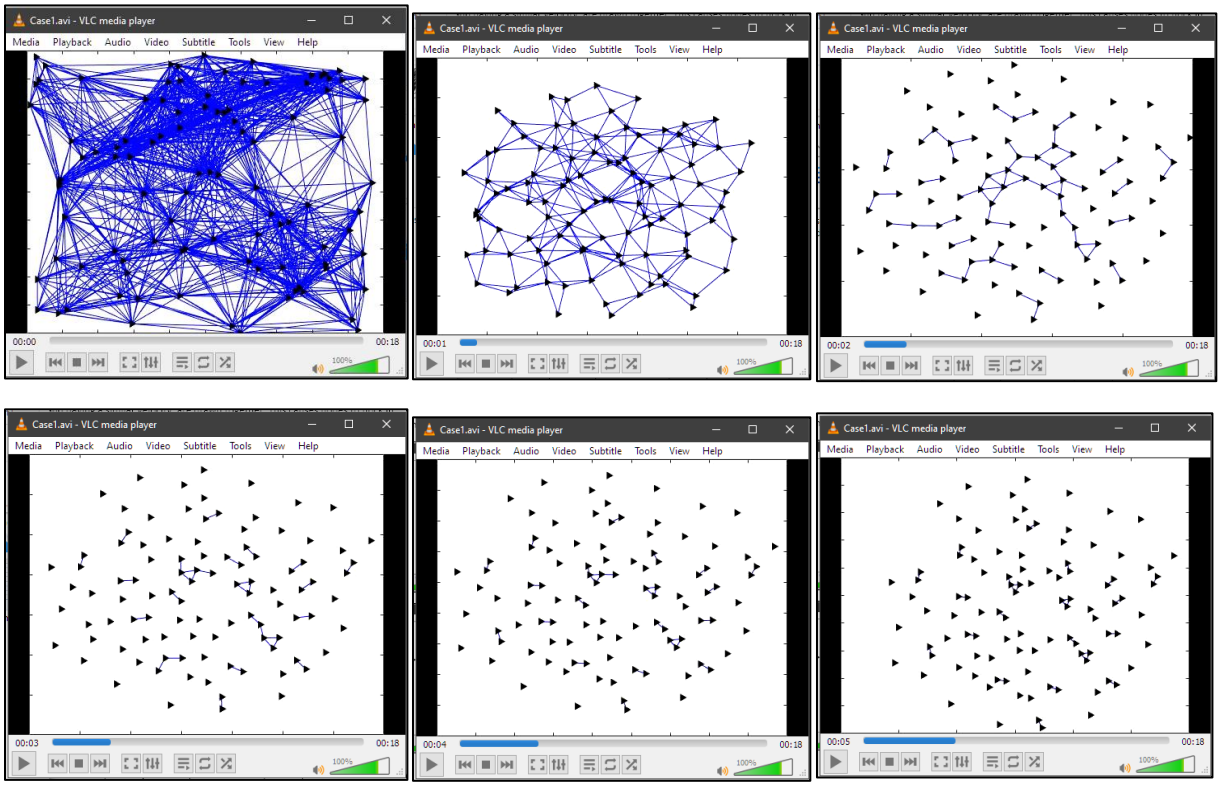
Components and Algorithms used:

To create a quasi-lattice, the nodes use a combination of forces to calculate their acceleration. These forces include a gradient-based term, a velocity consensus term, and a navigational feedback term. Each of these forces are weighted based on the 'C' values of our algorithm. The gradient term uses the term n_ij which creases a vector along the line connecting each node. The other component of the gradient determines the force of that vector, and whether the node needs to be pushed or pulled to reach the desired distance. The velocity consensus term uses a spatial adjacency matrix (a_ij) to find a node's neighbors, and then it matches its neighbor's velocity based on their proximity. The navigational feedback term is a simple norm returning the distance from a node to the target and it scales force according to how far away a node is from the target. Sigma norms are used to ensure vectors are differentiable, while a bump function is used to stop collisions. Overall, we are left with an algorithm that can create these flocking networks that flock towards a target location.

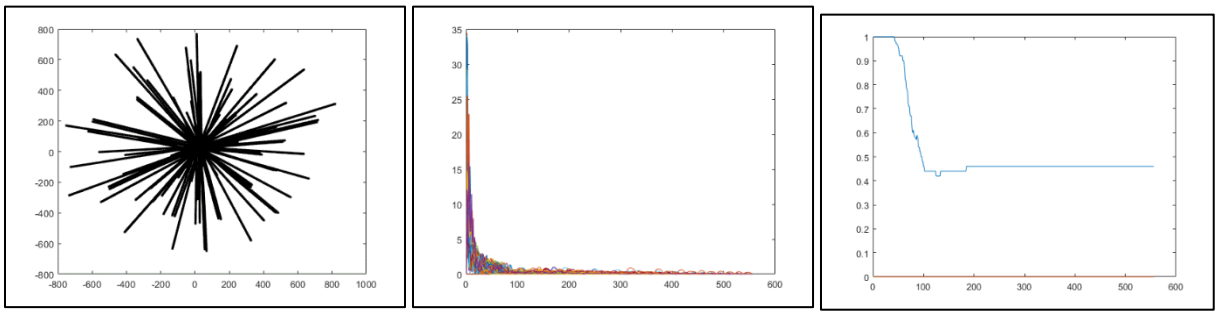


Case 1 - Algorithm 1:

Below are nine images from the MATLAB simulation. The first six images, starting from the top left, demonstrates how Algorithm 1 causes fragmentation. This fragmentation is caused by there being no navigational feedback term in the equation. The initial deployment in a 50x50 area causes the gradient forces to push against each other. Nodes that happen to be close, and having a similar velocity, are drawn together. This causes nodes to flock in small groups, but with no common goal, so they fly away from other small groups into space.

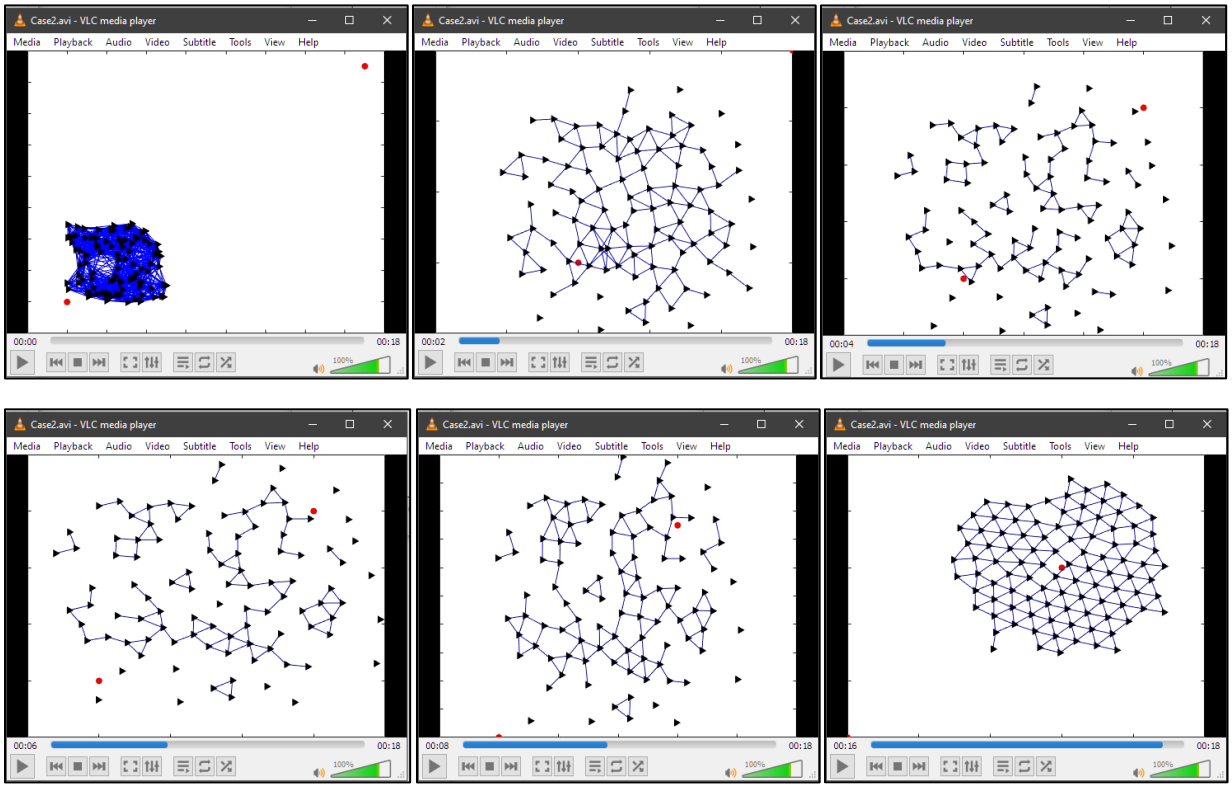


The next 3 images are plots of the nodes' trajectories, velocities, and connectivity. The expected result is for each node to fly off in every direction, which our trajectory plot accurately shows. The velocity graph looks to be as expected. Each line represents the velocity of each node. The velocity plot spikes initially, and then calms down after the nodes have stopped bumping into one another. The consensus and gradient forces find equilibrium, and thus the velocity of each node rests close to zero. The connectivity graph also acts as expected. The nodes start connected since they are in a 50x50 area, and then diverge from each other to avoid collisions. Once nodes start to flock together, they do not come apart, thus resting around 45% connectivity. This value can be increased through increasing the 'h' variable.

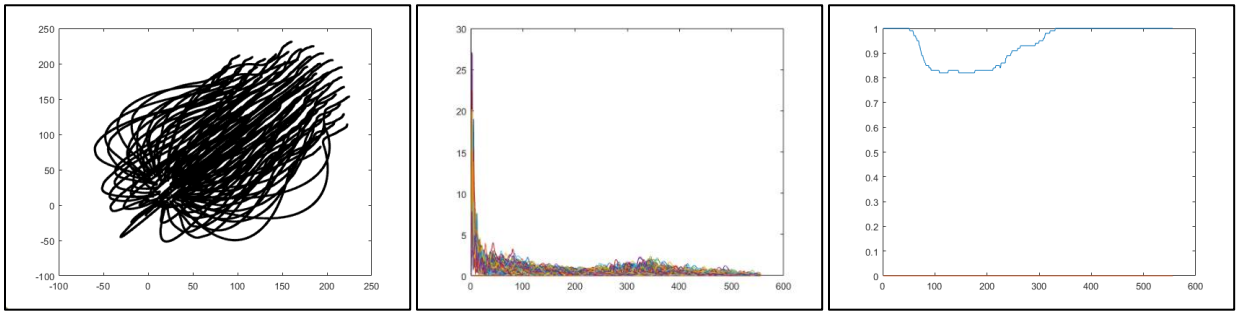


Case 2 - Algorithm 2 - Static Target:

The first six images, starting from the top left, demonstrates how Algorithm 2 tracks a static target. The nodes track this static target via the navigational feedback term. The initial deployment in a 50x50 area causes the gradient forces to push against each other. That is why the second image shows fragmentation. By the third image however, the nodes have started to flock to one another, and head toward the target. Eventually all the nodes link up to form a quasi-lattice, as shown in image 6.

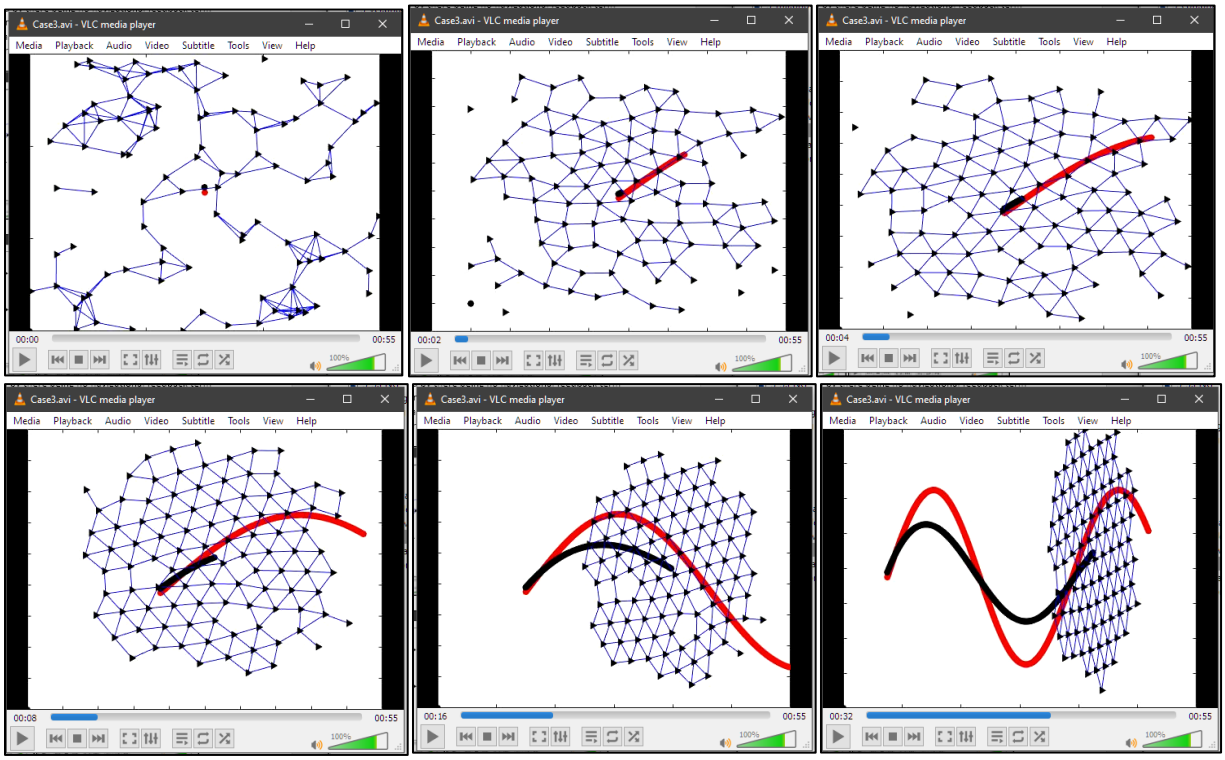


The next 3 images are plots of the nodes' trajectories, velocities, and connectivity. The expected trajectory is reflected in our trajectory plot. Each node moves towards the target eventually. The ones that do not immediately head to the target, are being forced away from their neighbors by the gradient. This is because tracking a target is the network's second priority in comparison to not bumping into other nodes. The velocity graph is similarly normal. My explanation for the two bumps in activity is that the nodes are initially moving towards the target causing a spike, and then the second bump comes from adjusting after the lagging nodes push the nodes already at the target forward. They then eventually reach equilibrium, reducing the velocity to zero. The connectivity graph shows that while the nodes start out fragmented, they came back together to create a connected network.

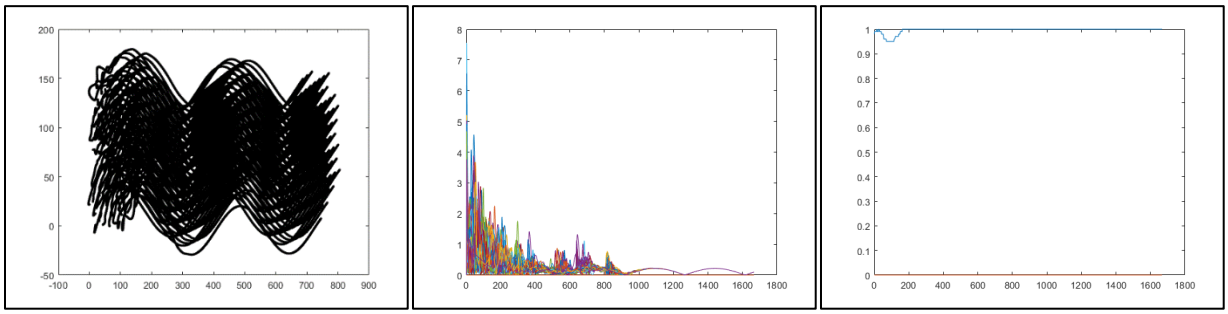


Case 3 - Algorithm 2 - Dynamic Target:

The first six images, starting from the top left, demonstrates how Algorithm 2 tracks a dynamic target. The effects of initial fragmentation are lessened this time, as we expanded the starting area to be 150x150, giving more room between nodes. This causes the nodes to form a quasi-lattice more quickly as shown in images 1 and 2. The main difference between a static and dynamic target, is that the there is an added component to the navigational feedback term. This component accounts for the target's acceleration and adjusts the navigational feedback's force to match. This causes the network to follow the dynamic target's sin wave pattern shown in images 5 and 6.

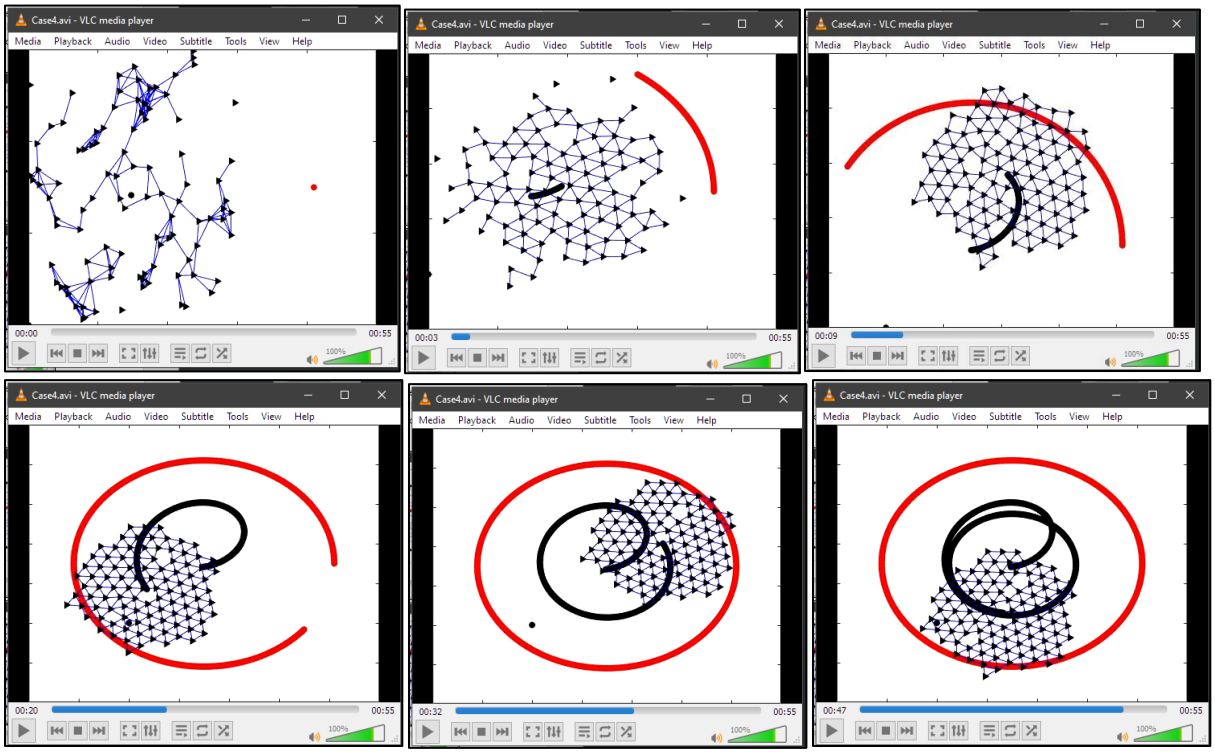


The next 3 images are plots of the nodes' trajectories, velocities, and connectivity. The trajectory graph, after forming a quasi-lattice, follows the sin wave trajectory of the target node. Each node individually follows this pattern, which becomes pertinent when analyzing the velocity graph. After the 900th iteration, notice that the velocity of every node follows the same pattern. This pattern correlates with the sin wave's period, showing that the navigational feedback term is working as intended. Likewise, the connectivity graph stays at 100% after the initial fragmentation.



Case 4 - Algorithm 2 - Circle Target:

The first six images, starting from the top left, demonstrates how Algorithm 2 follows a dynamic target following the path of a circle. Like the last case, the initial nodes are plotted in a 150x150 point area. Algorithm 2, with the parameters given, struggles with a circle target more than the sin wave. Image 6 demonstrates this because the center of mass (COM) lies within the target circle throughout case 4. Its performance could be improved by increasing the c_mt1 weight, which gives more priority towards the navigational feedback term. The user can test this for themselves by changing c_mt1 from 1.1 -> 3 and witnessing the results.



The next 3 images are plots of the nodes' trajectories, velocities, and connectivity. The trajectory plot demonstrates that the navigational feedback is working but doesn't tell the whole story. The velocity graph shows that eventually the velocity of every node equalizes like in case 3 thus showing algorithm 2 is working as intended. Finally, the connectivity graph remains at 100% after initial fragmentation.