Simulations of Multi-Waypoint Flocking Problem with Delays and Saturation

Taylor Johnson
GE525 - Spring 2009
May 15, 2009

***Abstract*—The flocking problem has been studied for some time, and is thought to be a good way to perform distributed control for swarms of multi-agent systems, such as UAVs. The majority of these studies have not placed real-world limitations on the distributed system that would be experienced. For example, delays and asynchrony due to wireless network transmission of state information and limited actuator operating range are infrequently analyzed. This project studies through simulation the flocking problem as a group of particles with double integrator dynamics moving from some initial condition towards multiple waypoints (one objective at a time), analyzing stability (convergence to a flock and convergence to the waypoints) while comparing a variety of real-world constraints: *a*) with and without state (velocity) saturation, *b*) with and without actuator saturation, *c*) synchronous versus asynchronous (with bounded delay) state feedback, and *d*) combinations of these. The future work section presents a hybrid system model being investigated for the formal analysis of these systems in the presence of another real-world constraint, failures.**

I. Introduction

Coordinated movement of multi-agent systems has been studied in detail for some time [1] [2], often with a focus on determining controls that lead to what is termed flocking [3] or cooperative behavior [4] [5]. Flocking behavior is defined as a group of agents with individual control laws eventually reaching an equidistant spacing between themselves, and is found in nature as in schools of fish [6], flocks of birds [7], particles [8], and more generally [9].

The work of Reynolds [10] established three general rules that are necessary for flocking behavior to emerge in these systems, *1*) "Collision Avoidance: avoid collisions with nearby flockmates" *2*) "Velocity Matching: attempt to match velocity with nearby flockmates" *3*) "Flock Centering: attempt to stay close to nearby flockmates". Looking at these rules, it is first necessary to analyze the term "nearby". For flocks and herds found in nature, nearby could be determined by the individual agents through sensory mechanisms, such as sight, sound, smell (maybe pheromones), and touch. For an engineered system, such natural sensory information could be determined from one agent to another by radar, laser radar, computer vision systems, etc. Alternatively, agents could tell one another over a wireless network about where they believe they are located, such as sharing information from global positioning system (GPS) state or cellular triangulation. Real systems would likely use a combination of these sensory mechanisms, but all interesting cases are dependent upon agents being physically near one another, so to this end, we consider a broad definition of nearby as agents being within some *communications radius*, r_{comm}, of one another.

Having defined nearby, the collision avoidance and flock centering rules of Reynolds necessitate some attractive and repulsive force between nearby agents that causes them to become equidistantly spaced. There are a variety of such forces that arise in nature, such as the Lennard-Jones potential, which is a combination of the attractive van der Waals force and the repulsive Pauli force between molecules [11] as seen in Figure 1. Velocity consensus is also necessary to maintain flocking behavior over time, and in addition, it has been shown in [3] that in a system without leaders, all nodes need to have a notion of what the goal or objective is to prevent regular fragmentation, as in their *Algorithm 2* which we will analyze in detail in Section II.

II. System Model

For the system model, we are closely following the work of [3]. The system is composed of N agents or nodes, each represented as a vertex in a set of vertices defined $V \subset \mathbb{N} = \{0 \ldots N\}$, of a graph $G := (V, \varepsilon)$, where the edges are defined as $\varepsilon \subseteq \{(i,j) : i,j \in V, j \neq i\}$. The state-space representation of node $i \in V$ is $\dot{q}_i = p_i$, $\dot{p}_i = u_i$, where $q_i, p_i, u_i \in \mathbb{R}^m$, and q_i represents position, p_i represents velocity, and u_i represents acceleration, or

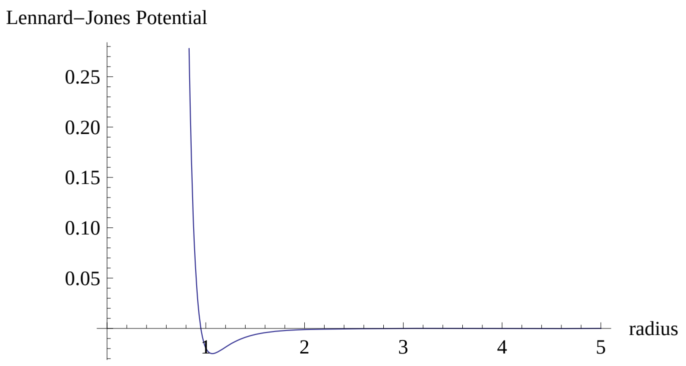

Fig. 1. Lennard-Jones Potential

proportionally, force. We denote the matrix of positions of all N agents as $q \in \mathbb{R}^{m\times N}$, velocities as $p \in \mathbb{R}^{m\times N}$, and controls as $u \in \mathbb{R}^{m\times N}$.

The notion of nearby as flockmates being within some communications radius of one another gives rise to a definition of the *communications neighbors* of agent i as $N_c(i,q) := \{j \in V : ||q_j - q_i|| < r_{comm}\}$. Each agent computes its own set of communications neighbors. Similarly, the definition of a flock as agents being equidistantly spaced gives rise to the definition of a flock as an α-*lattice* as the solutions of the set of constraints $||q_j - q_i|| = r_{lattice}, \forall j \in N_c(i,q)$. Alternatively, we could define the set of α-*lattice neighbors* as $N_l(i,q) := \{j \in N_c(i,q) : ||q_j - q_i|| = r_{lattice}\}$. This α-lattice definition of flocking is overly restrictive though and disallows any small perturbations in the system, so we consider a quasi α-lattice as $-\delta \leq ||q_j - q_i|| - r_{lattice} \leq \delta, \forall j \in N_c(i,q)$.

We do not need to consider any deviation from a perfect α-lattice in terms of some deviation energy, as this is required for stability proofs only, and is not used in the computation of an agent's control. Similarly, for real-world implementations in software we would not have such rigorously defined functions as in [3], so we will depart to some extent from the work of Olfati-Saber at this point, as for analysis purposes and a proof of convergence to a flock, they carefully constructs a continuous control. For simulations we can define the σ-norm ($\mathbb{R}^m \rightarrow \mathbb{R}_{\geq 0}$) simply as the Euclidean norm (2-norm), $||z||_\sigma := \sqrt{z_1^2 + \ldots + z_m^2}$, instead of $||z||_\sigma = \frac{1}{\epsilon}\left[\sqrt{1+\epsilon||z||^2} - 1\right]$, for $\epsilon > 0$. The difference is highlighted in Figure 2 and Figure 3. Note however that we do take the same values of r_α (here r^α_{comm}) and d_α (here $r^\alpha_{lattice}$) as though they were computed using their σ-norm, as otherwise it would be hard to compare results; these specific values are given in Section III.

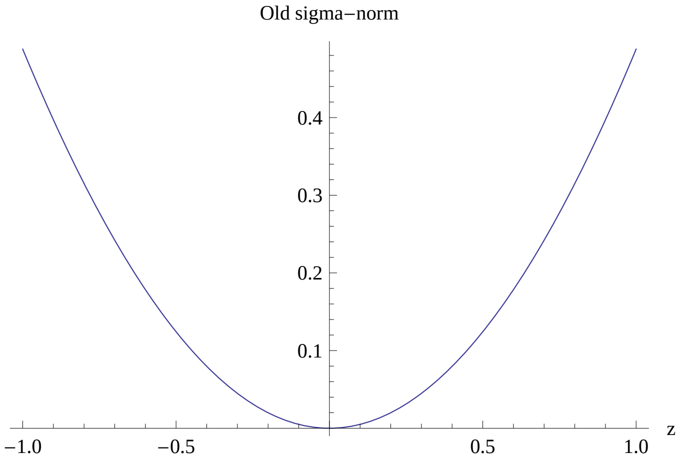

Fig. 2. Old σ-Norm

The gradient of the σ-norm, $\sigma_\epsilon(z) := \nabla||z||_\sigma$, is needed, and becomes $\sigma_\epsilon(z) := \frac{z}{||z||}$ instead of $\sigma_\epsilon(z) = \frac{z}{\sqrt{1+\epsilon||z||^2}} = \frac{z}{1+\epsilon||z||_\sigma}$. This is no longer differentiable at 0, but for simulations, this does not matter.

Similarly, the C^1-smooth bump function ρ_h can be redefined as

$$\rho_h(z) = \begin{cases} 1 & \text{if } z \in [0,h) \\ h - z & \text{if } z \in [h,1] \\ 0 & \text{else} \end{cases} \quad \text{instead of as}$$

$$\rho_h(z) = \begin{cases} 1 & \text{if } z \in [0,h) \\ \frac{1}{2}\left[1 + cos\left(\pi\frac{(z-h)}{(1-h)}\right)\right] & \text{if } z \in [h,1] \\ 0 & \text{else} \end{cases}.$$

These differences are seen in Figure 4 and Figure 5.

Using these new functions, we work with the same element-wise definition of the *spatial adjacency matrix*, $A := [a_{ij}]$, as Olfati-Saber, $a_{ij}(q) = \rho_h\left(\frac{||q_j - q_i||_\sigma}{r^\alpha_{lattice}}\right) \in [0,1], j \neq i$. We also use the same ϕ and ϕ_α functions, defined as $\phi(z) := \frac{1}{2}[(a+b)\sigma_1(z+c) + (a-b)]$ and $\phi_\alpha(z) := \rho_h(\frac{z}{r^\alpha_{lattice}})\phi(z - r^\alpha_{comm})$, where $\sigma_1(z) = \frac{z}{\sqrt{1+z^2}}$,

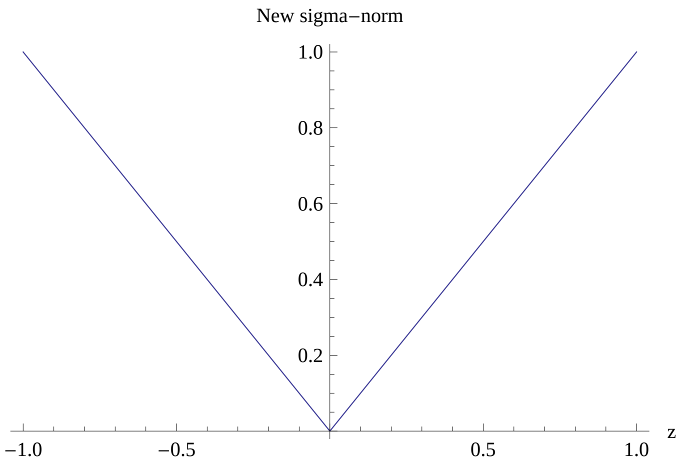

Fig. 3. New σ-Norm

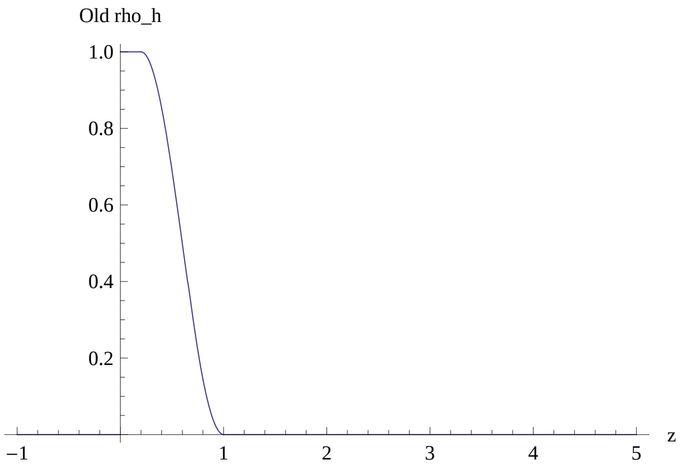

Fig. 4. Old ρ_h

$0 < a \leq b$, and $c := \frac{|a-b|}{\sqrt{4ab}}$. Note that while these are defined the same, both have changed due to them being composed of other functions that have changed. The change to ϕ is seen in Figure 6 and Figure 7 and to ϕ_α in Figure 8 and Figure 9.

We focus only on the control in the form of *Algorithm 2* presented in [3], which is defined as $u_i := u_i^\alpha + u_i^\gamma$, where

$$u_i^\alpha = \sum_{j \in N_c(i,q)} \phi_\alpha\left(||q_j - q_i||_\sigma\right) n_{ij} + \sum_{j \in N_c(i,q)} a_{ij}(q)(p_j - p_i)$$

and $u_i^\gamma = f_i^\gamma(q_i, p_i, q_r, p_r)$ with q_r and p_r representing the state of the goal, which may be changing, and where $n_{ij} := \frac{(q_j - q_i)}{\sqrt{1+\epsilon||q_j-q_i||^2}}$. The goal's dynamic state is modeled as a double integrator also, with state-space representation $\dot{q}_r = p_r$ and $\dot{p}_r = f_r(q_r, p_r)$. Note that we have taken $f_r(q_r, p_r) \equiv 0$, although it could be any function (and despite this, we set different waypoints at discrete instants in time).

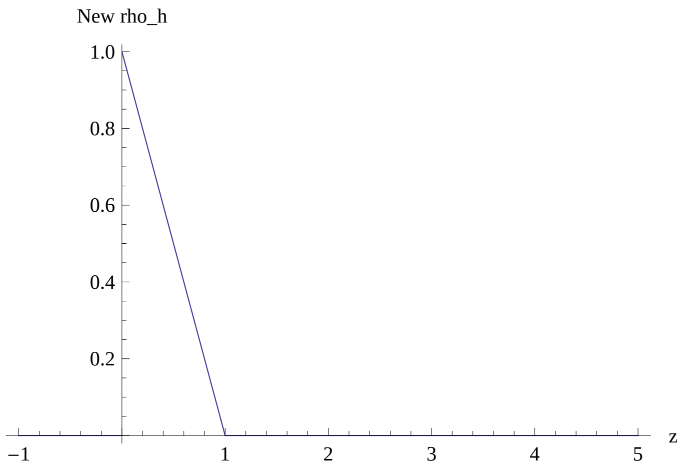

Fig. 5. New ρ_h

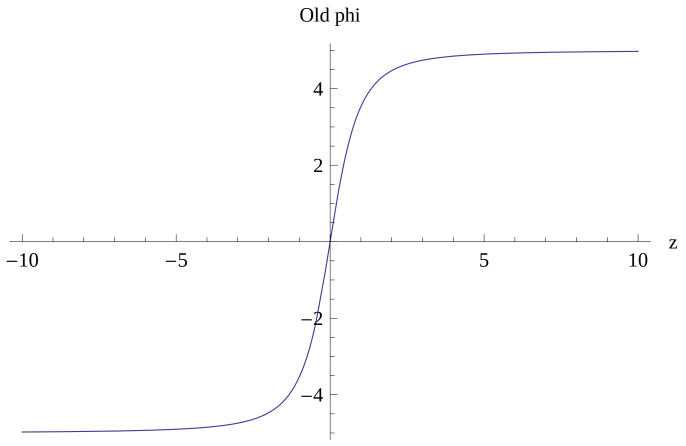

Fig. 6. Old ϕ

A. System Constraints

1) State and Actuator Saturation: There exists v_{max} and a_{max}, a maximum velocity and acceleration, respectively, such that, $\dot{q}_i \leq v_{max}$ and $\dot{p}_i \leq a_{max}$. The existence of a

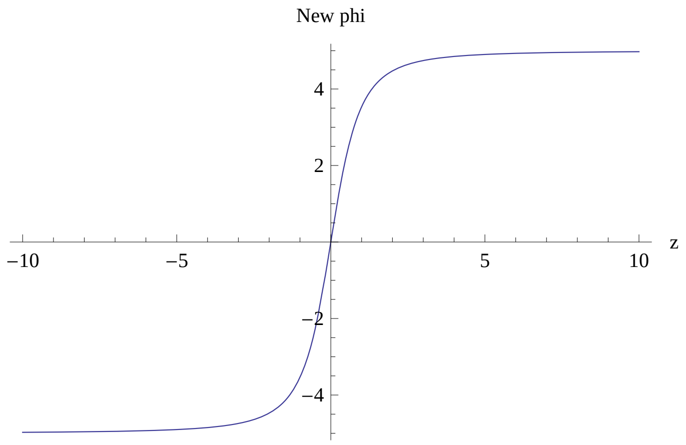

Fig. 7. New ϕ

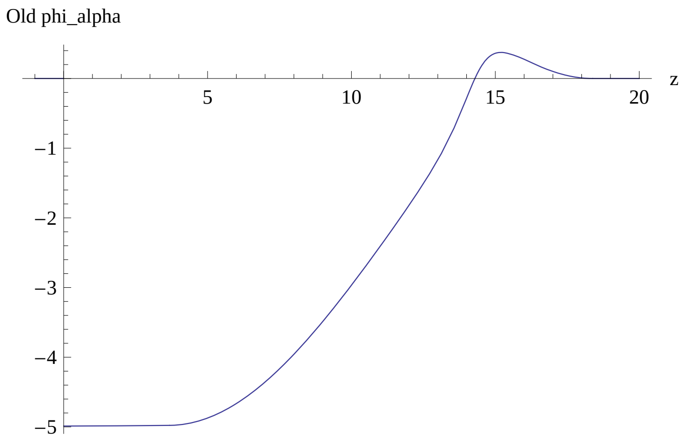

Fig. 8. Old ϕ_α

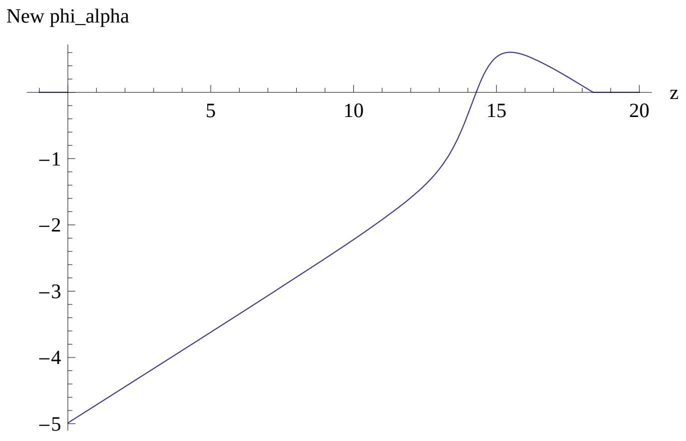

Fig. 9. New ϕ_α

maximum acceleration is reasonable given that it would be produced by some actuator with limited operating range and power source. Note that of course the existence of a maximum acceleration does not imply the existence of a maximum velocity. However, the existence of a maximum force that can be applied as a control, which is proportional to the maximum acceleration, and in the presence of opposing forces proportional to the velocity such that $\sum F = 0$ where $F \in \mathbb{R}^m$, does imply the existence of a maximum velocity, so this is a reasonable assumption. On Earth the balancing forces could be aerodynamic resistance, whereas in space, relativistic limits could be imposed.

For the simulations in Section III, we separately present results with state and actuator saturation. In general, actuator saturation is quite interesting, as it reduces any globally asymptotically stable system to a locally asymptotically stable system. Coupled with state constraints though, the problem is even more interesting, as not all of $\mathbb{R}^m$ is accessible by the states anyway, so all reachable states may be asymptotically stable nonetheless. These constraints were primarily motivated by previous work of ours on finding stabilizable regions by solving LMIs for an inverted pendulum system with limited actuator range, and with a limited set of safe states.

2) Synchronous and Asynchronous Communications with Delays:

III. Simulations

For all simulations, we work in 2 dimensions ($m = 2$) with $N = 50$ agents. We present one sample of 1-dimensional and 3-dimensional simulations in Figure **??** and Figure **??**, as the control and simulation environment work for m-dimensional systems. The sampling period is $T_c = 0.01$ seconds. Other constants used in the above definition of the controls are $a = 5$, $b = 5$, $c = X$,

A. Without Constraints

Before presenting results with constraints, we must first look at simulations without constraints for comparison. Figures 10 to 18 are a representative example, showing the system with 50 agents starting from random initial velocities in the range -100 to 100, at positions within 0 to 100, with a goal at qr=100, pr=20, c1=c2=0.1. The control value, separated into components, of one node over time can be seen in Figure **??**. The position value of all nodes as a function of time can be seen in Figure **??**; this plot can be particularly useful if one wants to verify more stringent properties, such as two agents never coming closer than some small amount within one another (of course this occurs with this control, but with a different setup, it may not).

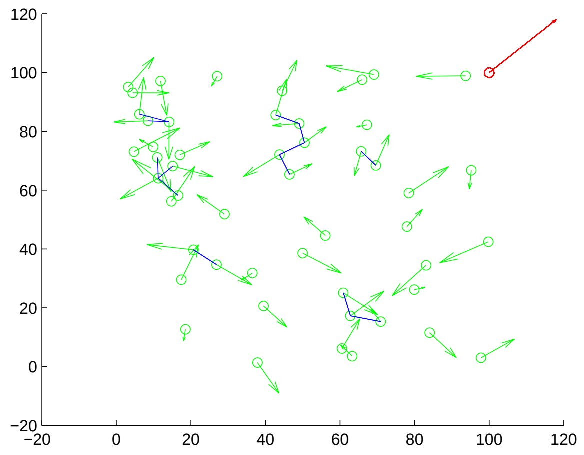

Fig. 10. N=50, m=2, Time=0s

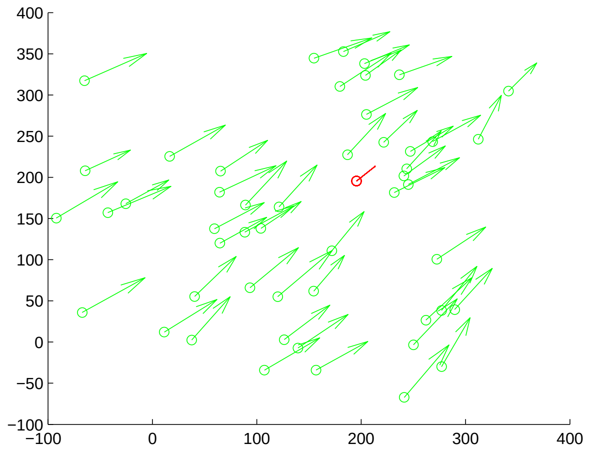

Fig. 11. N=50, m=2, Time=4s

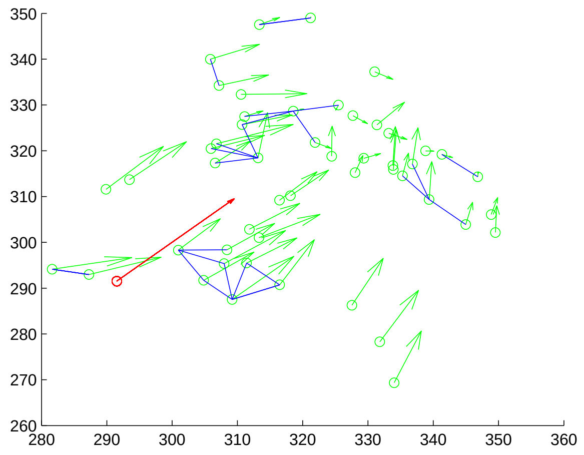

Fig. 12. N=50, m=2, Time=8s

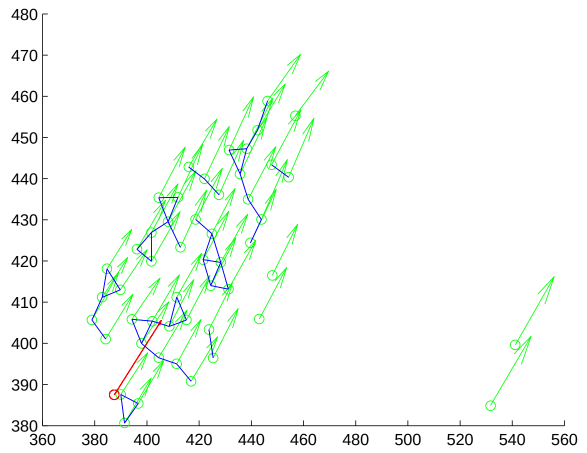

Fig. 13. N=50, m=2, Time=12s

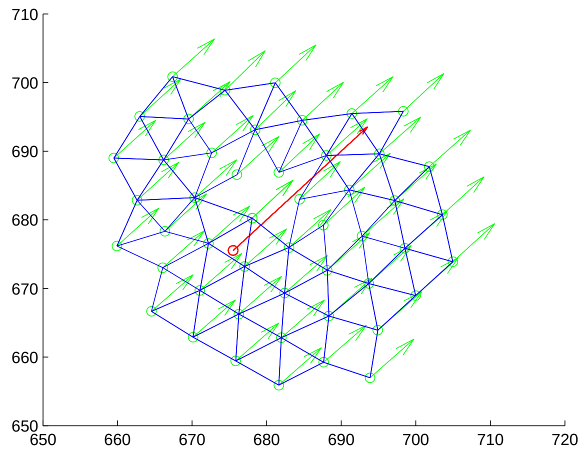

Fig. 16. N=50, m=2, Time=24s

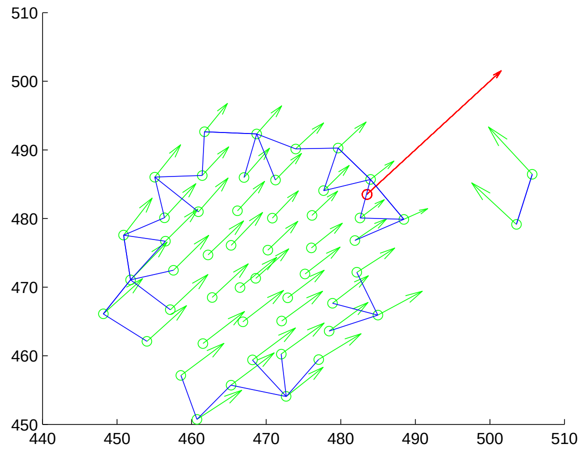

Fig. 14. N=50, m=2, Time=16s

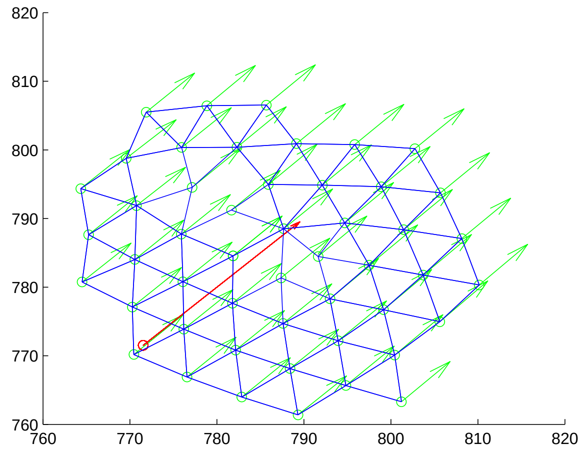

Fig. 17. N=50, m=2, Time=28s

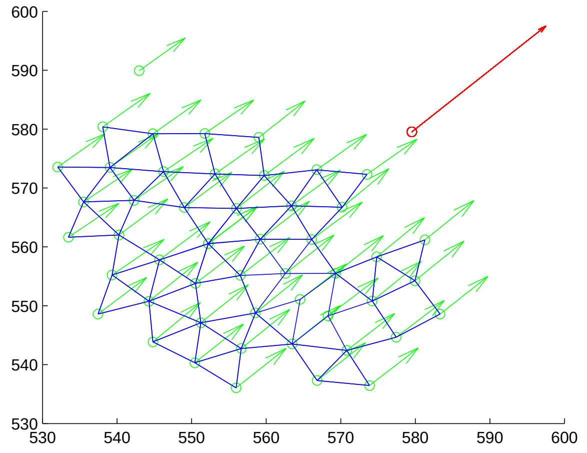

Fig. 15. N=50, m=2, Time=20s

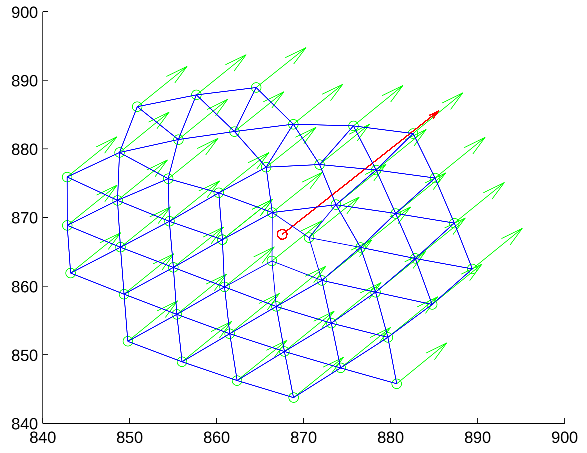

Fig. 18. N=50, m=2, Time=32s

B. State Constraints

Many simulations with constrained state were run, but for sake of presentation we show a comparison of a low maximum velocity ($v_max \equiv 10m/s$) and a high maximum velocity ($v_max \equiv 1e^6 m/s$).

C. Actuator Constraints

D. Communications Constraints

E. Combinations of Constraints

IV. Future Work

This work presented simulations of the effects that constraints that would be seen in real-world implementations of distributed control systems. However, our understanding of real-world distributed control systems would be incomplete without understanding the effect of failures on such systems. Computers and communication channels failures are captured by failure models from distributed computing such as crashes, omissions, and byzantine behavior, and their effect on computing systems have been well studied in the literature. In fact, the central theme in the distributed computing research is understanding the power and limitations of computing systems under different types of failures. Results in this area provide algorithms for solving canonical problems such as consensus, leader election, and clock synchronization in the presence of certain types of failures, and also establish lower bounds about impossibility of solving those problems with certain resource constraints. Distributed control systems consist not only of computers, but also the physical world which they interact with through sensors and actuators. Thus, to understand the power and limitations of real-world distributed control systems, we are initiating the study of such systems in the face of computer, communications, sensor, and actuator—failures. This will require new failure models, as sensors and actuators require completely new failure models, whereas crash, slow, and byzantine failure mitigation must be modified to ensure properties in the physical world. In addition, we need to consider new failure models for computers, such as timing failures. All of these failures now have physical manifestations, directly challenging safety of the control system.

In distributed computing, failure detection is often very hard, for example, one cannot tell differentiate between a slow and a crashed computer in an asynchronous system. In contrast, failure detection is often easy for actuators, for instance, if an aileron on an airplane is stuck-at a value, a sensor as simple as a potentiometer could detect that it is not responding to control action as it is not changing position. Timing failures can be analyzed offline through worst-case execution time (WCET) analysis, and also can be detected by runtime environment, such as in [12]. The prevalence of these classes of failures suggests new fault-tolerance strategies for distributed control systems. The overall goal is to combine fault detection with the model for physical manifestation of failure to avoid bad states.

Toward this goal, we are studying the hybrid system modeled as hybrid input/output automata (HIOA) [13]. The HIOA of the physical world and flocking algorithm are shown in the hybrid input/output automata language [**?**] in Figure 21 and Figure 22, respectively.

Informally, the system consists of N agents starting at arbitrary locations on the real line, where $N \in \mathbb{N}, N > 1$ with the goal of (a) reaching a fixed waypoint $x_g \in \mathbb{R}$, and (b) forming a *flock* or an equispaced formation on the way to x^*. The algorithm we shall study also works when a sequence of waypoints are presented to the agents, but for simplicity of presentation, we fix x_g.

The transition $send_i$ occurs at every time $\mathbf{x}.now = next_i$ and adds a message containing physical state measurements of $\mathsf{Agent_i}$ to a $Queue$ in $\mathsf{PhysicalWorld}$ for each of $\mathsf{Agent_i}$'s communication neighbors, to be delivered before $\mathbf{x}.now = d$, such that the state measurements are available for computation at $\mathbf{x}.now = \mathbf{x}.next_i$. Also, the control for $\mathsf{Agent_i}$ is set at this transition, based on the measurement values at the time $\mathbf{x}.now - \Delta$. The transition $recv_{ij}$ removes a message containing physical state measurements of $\mathsf{Agent_j}$ from a $Queue$ in $\mathsf{PhysicalWorld}$ which is for $\mathsf{Agent_i}$ and adds it to a *buffer*. Thus, each $\mathsf{Agent_i}$ receives knowledge of all nearby neighbors in time.

This state knowledge of neighbors is stored in a *buffer* which maps from the set of indices of nodes, $[N]$ to $(\mathbb{R} \cup \{\perp\})^3$. That is, if an $\mathsf{Agent_i}$ has not received a message from $\mathsf{Agent_j}$ yet, then $j \notin nbrs_i$, and thus the mapping with j will yield triple of $\{\perp\}$s. Each of the functions $\hat{x}$, $\hat{v}$, and $\hat{u}$ maps from the set of neighbors of $\mathsf{Agent_i}$, $nbrs_i$ to triples of physical state variables in R. This achieves the same result as accessing the *buffer*, we should clarify this, as it should be the case that $\hat{x}(j) = buffer(j) = x_j$ from time $\mathbf{x}.next_i - \Delta$. We are using the *buffer* behind the scenes to actually store the message values, and then just use the functions $\hat{x}$, $\hat{v}$, and $\hat{u}$ as a convenient notation. TODO: There is some confusion here–why are we setting the buffer and hat functions separately?

The control, $ctrl(i, x_g, x_i, v_i, u_i, \hat{x}, \hat{v}, \hat{u}, nbrs_i)$, is defined as

where $\tilde{x} = \hat{x}(\tilde{i})$, $\tilde{v} = \hat{v}(\tilde{i})$, $\tilde{u} = \hat{u}(\tilde{i})$, $\tilde{i} = \operatorname*{argmin}_{j \in nbrs_i} (\hat{x}(j) - x_i)$, $a \in \mathbb{R}_+$, and $b \in \mathbb{R}_+$.

Let $\mathcal{A}$ be the HIOA obtained by composing $\mathsf{PhysicalWorld} \| \mathsf{Agent}_1 \| \ldots \mathsf{Agent}_N$. We denote a state of $\mathcal{A}$ by $\mathbf{x}$ and individual state components by $\mathbf{x}.x_i$, $\mathbf{x}.next_i$, etc.

We would like to show that the *safety property*, that the distance between each node i and j is at least r_{safety} is satisfied. Assume that the network delay between nodes is bounded, such that a message sent by node i is received by all its communication neighbors within time d. This assumption immediately gives that all $recv_{ij}$ and $recv_{ji}$ actions will be correct, in that the necessary state information is delivered before the next period.

Invariant IV.1. $\mathcal{I}_{safety}(S) =$

$$\begin{cases} \text{if } (\mathbf{x}.now = \mathbf{x}.next_i) \wedge \neg \mathbf{x}.Send \Rightarrow \\ \forall i \left(\mathbf{x}.x_i - \mathbf{x}.x_{i-1}\right) + \Delta\left(\mathbf{x}.v_i - \mathbf{x}.v_{i-1}\right) + \frac{1}{2}a_{max}\Delta^2 \geq \\ r_{safety} \\ \text{if } (\mathbf{x}.now \neq \mathbf{x}.next_i) \vee (\mathbf{x}.now = \mathbf{x}.next_i \wedge Send) \Rightarrow \\ \forall i \left(\mathbf{x}.x_i - \mathbf{x}.x_{i-1}\right) + t_{Rem}\left(\mathbf{x}.v_i - \mathbf{x}.v_{i-1}\right) + \frac{1}{2}a_{max}t_{Rem}^2 \geq \\ r_{safety} \end{cases}$$

where $t_{Rem} \equiv (\mathbf{x}.next_i - \mathbf{x}.now)$.

V. Conclusions

References

[1] W. M. Spears, D. F. Spears, J. C. Hamann, and R. Heil, “Distributed, physics-based control of swarms of vehicles,” *Auton. Robots*, vol. 17, no. 2-3, pp. 137–162, 2004.

[2] D. M. Stipanovic, G. Inalhan, R. Teo, and C. J. Tomlin, “Decentralized overlapping control of a formation of unmanned aerial vehicles,” *Automatica*, vol. 40, no. 8, pp. 1285 – 1296, 2004.

[3] R. Olfati-Saber, “Flocking for multi-agent dynamic systems: algorithms and theory,” *Automatic Control, IEEE Transactions on*, vol. 51, no. 3, pp. 401–420, March 2006.

[4] J. Fax and R. Murray, “Information flow and cooperative control of vehicle formations,” *Automatic Control, IEEE Transactions on*, vol. 49, no. 9, pp. 1465–1476, Sept. 2004.

[5] A. Ryan, M. Zennaro, A. Howell, R. Sengupta, and J. Hedrick, “An overview of emerging results in cooperative uav control,” vol. 1, Dec. 2004, pp. 602–607 Vol.1.

[6] E. Shaw, “Fish in schools,” *Natural History*, vol. 84, no. 8, pp. 40–45, 1975.

[7] B. L. Partridge, “The chorus-line hyopthesis of maneuver in avian flocks,” *Nature*, vol. 309, pp. 344–345, 1984.

[8] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet, “Novel type of phase transition in a system of self-driven particles,” *Phys. Rev. Lett.*, vol. 75, no. 6, pp. 1226–1229, Aug 1995.

[9] A. Okubo, “Dynamical aspects of animal grouping: Swarms, schools, flocks, and herds,” *Adv. Biophys.*, vol. 22, pp. 1–94, 1986.

[10] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” in *SIGGRAPH ’87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1987, pp. 25–34.

[11] J. E. Jones, “On the determination of molecular fields. ii. from the equation of state of a gas,” *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 106, no. 738, pp. 463–477, 1924. [Online]. Available: http://www.jstor.org/stable/94265

[12] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, “Giotto: A time-triggered language for embedded programming,” in *Proceedings of the IEEE*. Springer-Verlag, 2001, pp. 166–184.

[13] N. Lynch, R. Segala, and F. Vaandrager, “Hybrid i/o automata,” *Inf. Comput.*, vol. 185, no. 1, pp. 105–157, 2003.

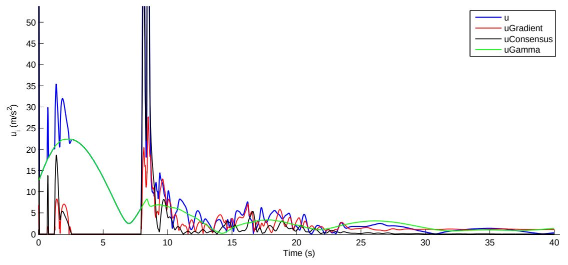

Fig. 19. N=50, m=2, Control Value of One Node over Time

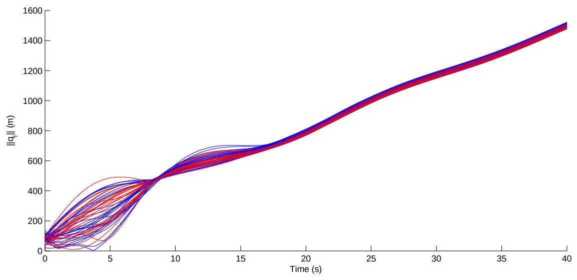

Fig. 20. N=50, m=2, Position Values of All Nodes over Time

```
automaton PhysicalWorld($d$ : Real$^+$, $r_{comm}$ : Real$^+$, $r_{init}$ : Real$^+$)
  where $d < \Delta$
  signature
    input send$_i$($x, v, u$ : Real)
    output recv$_{ij}$($x, v, u$ : Real)

  variables
    output $x_i, x_g, v_i$ : Real$\forall i \in [N]$;
    input $u_i$ : Real$\forall i \in [N]$;
    internal $Channel_{ij}$ : $Queue[[< msg$ : Real$^3, dl$ : Real$^+ >]] := empty$;
          $now$ : Real := 0;
    initially $|x_i - x_j| \geq r_{init} \forall i, j \in [N], i \neq j$;
 $x_g = +\infty$;

  transitions
    input send$_i$($x, v, u$)
      eff: $Channel_{ij} := append(Channel_{ij}, << x, v, u >, now + \Delta_{delay})$,
 $\forall j : |x_i - x_j| \leq r_{comm}$;

    output recv$_{ij}$($x, v, u$)
      pre $\exists! t \in$ Real $:< x, v, u, t >\in Channel_{ij}$
      eff $Channel_{ij} = Channel_{ij} - \{< x, v, u, t >\}$;

  trajectories
    trajdef hold
      evolve $\forall i \in [N]$:
          $d(x_i) = v_i$;
          $d(v_i) = u_i$;
          $d(now) = 1$;
      stop when $\exists i, j \exists m, t :< x, v, u, t >\in Channel_{ij} \wedge t = now$;
```

Fig. 21. HIOA Model Physical World

```
automaton $\mathsf{Agent_i}(\Delta : \texttt{Real}, i : [N])$ where $\Delta > 0$
  signature
    input $\mathsf{recv_{ji}}(x, v, u : \texttt{Real})$
    output $\mathsf{send_i}(x, v, u : \texttt{Real})$

  variables
    input $x_i, x_g, v_i : \texttt{Real}$;
    output $u_i : \texttt{Real} := 0$;
    internal $nbrs_i \subseteq [N]$;
            $\hat{x}, \hat{v}, \hat{u} : nbrs_i \mapsto \texttt{Real}$;
            $now_i : \texttt{Real} := 0$;
            $next_i : \texttt{Real} := \Delta$;
            $buffer : [N] \mapsto (\texttt{Real} \cup \{\perp\})^3$;

  transitions
    input $\mathsf{recv_{ji}}(x, v, u)$
      eff $buffer[j] =< x, v, u >$;
          $nbrs_i = nbrs_i \cup \{j\}$;
          $\hat{x}[j] = x$;
          $\hat{v}[j] = v$;
          $\hat{u}[j] = u$;

    output $\mathsf{send_i}(x, v, u)$
      pre $now_i = next_i \wedge x = x_i \wedge v = v_i \wedge u = u_i$;
      eff $next_i = next_i + \Delta$;
          $u_i := ctrl(i, x_g, x_i, v_i, u_i, \hat{x}, \hat{v}, \hat{u}, nbrs_i)$;
          $nbrs_i = \{\}$;
          $sent_i = T$;

  trajectories
    trajdef hold
      evolve $d(now_i) = 1$;
      stop when $now_i = next_i$;
```

Fig. 22. HIOA Model of $Agent_i$