# Project: Simulations of Multi-Objective 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 of swarms of unmanned autonomous vehicles (UAVs) [?]. In general though, these studies have not placed realistic limitations on the system that would be experienced, such as delays and asynchrony due to wireless network transmission and limited actuator operating range. This project studies through simulation the flocking problem as a group of particles with double integrator dynamics as in [1] moving from some initial condition towards multiple objectives, analyzing stability (convergence to a flock and convergence to the goals) while comparing a variety of combinations of real world constraints: a) with and without state saturation, b) with and without actuator saturation, and c) synchronous versus asynchronous with bounded delay state feedback.**

## I. Introduction

a. What is flocking

b. Dynamics: double integrator, Olfati-Saber controller model

Our understanding of real-world computing systems would be incomplete without understanding the effect of failures on such systems. Computers and communication channels failures are captured by failure models 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, in this paper, we initiate the study of such systems in the face of computer, sensor, and actuator—failures.

Rough notes: New failure models and their physical manifestations. Sensors and actuators require completely new failure models. Crash, stuck-at, byzantine. Give examples. Also need to consider new failure models for computers, e.g. timing failures. These failures now have physical manifestations, directly challenging safety of the control system.

Failure detection. In distributed computing failure detection is often very hard. E.g., we cannot tell differentiate between a slow and a crashed computer in an asynchronous system. In contrast, failure detection is often easy for actuators. Give example. Timing failures can be detected by runtime environment.

The above suggests new fault-tolerance strategies for distributed control systems. Mantra: Combine fault detection with the model for physical manifestation of failure to avoid bad states.

Our approach (this is very rough): Middleware detects failures and presents a modified world view to the controller. In this world view, faulty agents are "obstacles". Controller reacts to obstacles in the "usual" way.

Case studies. Flocking. Coverage.

Organization:

## II. Background

Brief description of HIOA. Executions, invariants, etc.

This material is likely to be used by both the flocking and the coverage case studies.

Let us describe the physical world model of the two case studies here. The two physical worlds are similar in that they both model the channels (Communications radius: $r_{comm}$), but they differ in the physics of agents.

We describe the failure models for sensors and actuators here. These failure models are also similar for the two case studies.

Denote the communication radius as $r_{comm}$, the initial radius to maintain safety as $r_{init}$, and the safety radius as $r_{safety}$. To maintain safety, we must have that $r_{comm} > r_{init} > r_{safety}$. Denote the time required to change a node's velocity from maximum to minimum (and vice-versa) as $t_a = \frac{2v_{max}}{a_{max}}$. Denote the time required to travel between the communication radius and the safety radius as $t_v = \frac{r_{comm}-r_{safety}}{2v_{max}}$. Then, $\Delta \leq \min(t_a, t_v)$. Denote the distance covered while changing a node's velocity from one extrema to the other as $r_a = 2v_{max}t_a = \frac{4v_{max}^2}{a_{max}}$. Denote the maximum distance traveled during the delay as $r_d = 2v_{max}\Delta$. (Note that we must fix one of $r_{comm}$ or $\Delta$, as they are functions of one another. We can then ensure the given $r_{comm}$ and computed $\Delta$ (or vice-

**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** $Channel_{ij} : Queue[[< msg : \text{Real}^3, dl : \text{Real}^+ >]] := empty$;
        $now$ : 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. 1. HIOA Model Physical World

versa) satisfy the initial conditions.) Now explicitly, define $r_{init} \equiv r_{safety} + r_a + r_d$, and require $r_{comm} \geq r_{init} + r_d$.

We assume the following of $S_0$:

1) $\mathbf{x}.x_i > 0$
2) if $j > i \Rightarrow \mathbf{x}.x_j > \mathbf{x}.x_i$
3) $|\mathbf{x}.v_i - \mathbf{x}.v_j| < 2v_{max}$
4) $|\mathbf{x}.x_i - \mathbf{x}.x_j| > r_{init} \qquad \forall i, j \in N$ and $i \neq j$.
5) $u_i = 0$
6) $x_g > x_i$
7) $\Delta \leq \min(t_a, t_v)$

## III. System Constraints

## IV. Simulations

## V. Flocking Analysis

In this section, we study a distributed flocking algorithm under several failure models. 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 $x^* \in \mathbb{R}$, 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 $x^*$.

### A. Specification

The flocking algorithm is specified by the HIOA Agent$_i$ of Figure 2.

Describe the code. State variables, actions, transitions, trajectories.

The transition $send_i$ occurs at every time $\mathbf{x}.now = next_i$ and adds a message containing physical state measurements of Agent$_i$ to a $Queue$ in PhysicalWorld for each of 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 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 Agent$_j$ from a $Queue$ in PhysicalWorld which is for Agent$_i$ and adds it to a $buffer$. Thus, each 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 Agent$_i$ has not received a message from 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 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?

**automaton** Agent$_i(\Delta$ : Real$, i : [N])$ **where** $\Delta > 0$
  **signature**
    **input** recv$_{ji}(x, v, u$ : Real$)$
    **output** send$_i(x, v, u$ : Real$)$

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

  **transitions**
    **input** 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** 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. 2. HIOA Model of $Agent_i$

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

$$ctrl\,(i, x_g, x_i, v_i, u_i, \hat{x}, \hat{v}, \hat{u}, nbrs_i) =$$

$$\begin{cases} \operatorname{sgn}(x_g - x_i)\, a_{max} & \text{if } nbrs_i = \{\} \\ \begin{cases} \operatorname{sgn}(\tilde{v})\, a_{max} \text{ if } |x_i - \tilde{x}| \leq r_{init} \\ a(\tilde{x} - x_i - r_{init}) + b(\tilde{v} - v_i) \text{ else} \end{cases} & \text{else} \end{cases}$$

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

### B. Analysis

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. However, before we begin working toward the safety property, we must make some assumptions about communications and prove some simpler invariants.

Assume that the network delay between nodes is bounded, such that a message sent by node $i$ is received by all its communication neighbors $j \in N_c(i, s)$ 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.

Lets distinguish between state and physical location, velocity etc. Because the state of each agent includes other variables such as $nbrs_i$, etc.

Recall from above that $d < \Delta$, that is, the *communications delay* $d$ is such that messages are delivered before control updates that occur at $\Delta$.

**Invariant V.1.** $\forall i, j, \mathbf{x}.now_i = \mathbf{x}.now_j = \mathbf{x}.now$.

The next invariant follows from synchronous communication. All nodes receive messages about the state information from time $next - \Delta$ before time $next$. Define the time for the last send with respect to state $s$ as $start(s) = s.next - \Delta$.

**Invariant V.2.** $\mathbf{x}.nbrs_i = N_c(i, \mathbf{x})(start(\mathbf{x}))$

$\mathcal{I}_{nbrs}(S) =$

$$\begin{cases} \text{if } \mathbf{x}.now - start(\mathbf{x}) \geq d \\ \text{then } \mathbf{x}.nbrs_i = N_c(i, \mathbf{x})(start(\mathbf{x})) \\ \text{elseif } \mathbf{x}.now - start(\mathbf{x}) < d \\ \text{then } N_c(i, \mathbf{x})(start(\mathbf{x})) = \mathbf{x}.nbrs_i \cup \{\mathbf{x}.Channel_{ij}\} \end{cases}$$

*Proof:* Fix a reachable state $s$. Expanding $s.now - start(s) \geq d$ to $s.now - s.next + \Delta \geq d$, it is clear that if $s.now \geq start(s) + d$, all messages will have been delivered due to the stopping condition for the $recv_{ij}$ actions to occur in $PhysicalWorld$. However, before that time in the other case, all messages may still be in the channel since no stopping condition has been reached (and all messages were put in the channel at time $start(s)$), or some messages may have been delivered already, so all messages are either in the channel or have been delivered, and thus, the set of neighbors of node $i$ is the union of these. All closed trajectories $\tau$ that could violate this are excluded by the stopping condition. ■

**Invariant V.3.** $\mathcal{I}_{channelSize}(S) = size(\mathbf{x}.Channel_{ij}) \leq 1$

*Proof:* This holds trivially since all messages will be delivered by time $start + d$, so they will be inserted and removed from the queue in $PhysicalWorld$ before the next $\Delta$. The size when $\mathbf{x}.now = start(\mathbf{x})$ is 1, since all messages are in the channel after the $Send_i$ action occurs at time $start(\mathbf{x})$. The size between $\mathbf{x}.now = start(\mathbf{x})$ and $\mathbf{x}.now = start(\mathbf{x}) + d$ is either 0 or 1, since some messages may have been delivered, but some may not have. The size between $\mathbf{x}.now = start(\mathbf{x}) + d$ and $\mathbf{x}.next$ is 0, since all messages must have been delivered by the stopping conditions and thus no closed trajectory $\tau$ could have crossed this boundary. ■

**Invariant V.4.** The set of *communication neighbors* of node $i$ at time $next - \Delta$ is the same as the set of *physical neighbors* at time $start + d$.

Assume $S_0$ is given such that $\mathcal{I}_{recv}(S_0)$ is satisfied at time $t_0$, then $\mathcal{I}_{recv}(S)$ is satisfied, and thus the receive property is maintained $\forall t \geq t_0$.

*Proof:*

This statement can be made more precise. But the property you want, I think, is the following invariant: Let $\mathbf{x}$ be a reachable state of the complete system.

For every agent i, j, if (a) $\mathbf{x}.now_i = \mathbf{x}.next_i$ (i.e., process i is about to send), and (b) $|\mathbf{x}.x_i - \mathbf{x}.x_j| <= r_{comm} - 2v_{max} * \Delta$ (i and j are "close") then $j \in \mathbf{x}.nbrs_i$ (i knows about j).

As usual, you may have to strengthen this invariant in order to make it inductive. This statement does not say anything about $\mathbf{x}.now_i! = \mathbf{x}.next$, you'll probably need to say something about that.

If $r_{comm} > r_{init}$, then nodes within at least $r_{comm}$ of one another get messages before their distance is less than $r_{init}$. Nodes may then be within between $r_{init}$ and $r_{safety}$ of one another.

We are given that $S_0$ is such that at the initial time $t_0$ the invariant $\mathcal{I}_{recv}(S_0)$ is satisfied. This gives us the following

$\forall S \xrightarrow{a} S' : \mathcal{I}_{recv}(S) \Rightarrow \mathcal{I}_{recv}(S')$

We consider each action separately in cases:

i) $send_i$

ii) $recv_i j$ If the delay $d$ between messages is less than the period of synchronous communication $\Delta$, then all $Recv$ actions will have occurred before the next period.

$\forall \tau$, if $\forall S \xrightarrow{\tau} S' : \mathcal{I}_{recv}(S) \Rightarrow \mathcal{I}_{recv}(S')$

While it is tempting to say that the worst scenario for $Recv$ is when two neighbors $i$ and $j$ are moving away from one another with maximum velocity and acceleration, this is actually not the case if we only care about the safety property defined as no nodes come within $r_{safety}$ of one another. The worst case is still when these two nodes are coming together with maximum

acceleration and velocity, and that they are **not** yet within $r_{comm}$. If they can cover their distance $|x_i - x_j|$ before the next $\Delta$, plus some details on the velocity and number of cycles required to change direction, then they may collide. What we want is an upper bound of $\Delta$ to guarantee everything, such as the following:

Note that we really should not have to talk about $u_i > 0$ as different to the $a_{max}$ case, since the worst affect on $r_{safety}$ comes from $v_i$, which if bounded by $v_{max}$ is already done. Any time we have acceleration and the particles are moving at less than $v_{max}$, as long as we know we work for $v_{max}$, we will work for anything less than $v_{max}$, regardless of the rate of change of $v_i$.

However, when two nodes are moving away from one another with maximum acceleration and velocity is the worst case for flocking. If we enforce an initial condition about all nodes being within $r_{comm}$, and that their initial velocities are not such that they can leave $r_{comm}$ before the first control update, then we should be able to guarantee flocking for all time. This gives us both an upper and lower bound on $r_{comm}$. (Note: We should check to see if this definition on initial conditions is basically what Olfati-Saber is referring to in the previous work for Algorithm 1 to maintain flocking instead of segmentation.)

■

**Invariant V.5.** $I_{reorder}(S) = \mathbf{x}.x_i < \mathbf{x}.x_j \forall i < j$

*Proof:* The base case is satisfied by assumption on the initial conditions that $\mathbf{x}.x_i < \mathbf{x}.x_j$.

The inductive case requires using the assumptions on the initial conditions are such that using the specified control maintains this ordering property. ■

**Invariant V.6.** $\mathcal{I}_{safety}(S) =$

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

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

*Proof:* There are three possible cases that can arise for the controls at $\mathbf{x}.next_i$, assuming at $start(\mathbf{x}, i)$ the state is okay: There are three interesting cases, two of which are when nodes can now communicate when they could not before, and the other is when nodes are still communicating. If nodes are not communicating, everything is boring as they just apply the go-to-goal control.

- First, when nodes are still more than $r_{init}$ apart, they apply $r_{comm} \geq |x_i - x_j| \geq r_{init} > r_{safety}$ $u_i = a \, (\tilde{x} - x_i - r_{init}) + b \, (\tilde{v} - v_i)$
- Second, when nodes are less than $r_{init}$ apart, they apply the collision avoidance control $r_{comm} \geq r_{init} > |x_i - x_j| > r_{safety}$ $u_i = \text{sgn} \, (v_j) \, a_{max}$ The only time this situation arises (as in, this is the only time we can recover from it) is as follows: $r_{comm} \geq r_{init} > |x_i - x_j| > r_{init} - r_d > r_{safety}$ that is, the nodes must still be separated by almost the radius required to recover safety, $r_{init}$, but can tolerate the distance covered over the delay.
- If nodes could previously communicate and still can, it is an interesting case, as they could be in either of the above cases. If $r_{comm} \geq |x_i - x_j| \geq r_{init} > r_{safety}$, then $u_i = a \, (\tilde{x} - x_i - r_{init}) + b \, (\tilde{v} - v_i)$. Otherwise, if $r_{comm} \geq r_{init} > |x_i - x_j| > r_{safety}$ $u_i = \text{sgn} \, (v_j) \, a_{max}$
- The most boring case is when nodes are still not communications neighbors, they just go towards the goal. $|x_i - x_j| > r_{comm} > r_{init} > r_{safety}$ $u_i = \text{sgn} \, (x_g - x_i) * a_{max}$

Assume $S_0$ is given such that $\mathcal{I}_{safety}(S_0)$ is satisfied. If $\mathcal{I}_{safety}(S_0)$ is satisfied, then $\mathcal{I}_{safety}(S)$ is satisfied, and thus the safety property is maintained.

We are given that $S_0$ is such that at the initial time $0$ the invariant $\mathcal{I}_{safety}(S_0)$ is satisfied. This gives us the following

$\forall S \xrightarrow{a} S' : \mathcal{I}(S) \Rightarrow \mathcal{I}(S')$

We consider each action separately in cases: i) $send_i$ Send does not affect the state of node $j$, and it does not affect the state of node $i$ until $next$, so it is vacuously true in both cases.

ii) $recv_i j$ Trivially from the $I_{recv}$ invariant.

$\forall \tau, if \forall S \xrightarrow{\tau} S' : \mathcal{I}(S) \Rightarrow \mathcal{I}(S')$

For the trajectories, it is sufficient to consider the worst possible trajectory affecting the safety condition, that is, when particles are coming towards one another with maximum acceleration. Given the assumption that $|x_i - x_j| < C_1$, we see that the relative velocities of any particles are bounded, and we will be immediately finished.

In reality, given a large enough $r_{comm}$ (or small enough $\Delta$), this is overly conservative, as the control law would prohibit nodes from moving towards one another with acceleration $a_{max}$. ■

**Invariant V.7.** Progress

All nodes reach the goal.

*Proof:* Show that if $x_g > x_i$, then $v_i > 0 \forall i$ eventually and for long enough time to reach goal. Symmetrically, if $x_g < x_i$, then $v_i < 0 \forall i$ eventually and for long enough time to reach goal. That is, we must show that $\exists t_g$ such that $v_i t_g \geq x_g \forall i$. (Slightly more complicated than this, integral of velocity over time?)

Is it useful to do this proof as a stability proof, showing that eventually, $\forall i, v_i = v_max$, in the direction of the goal?

Or equivalently, showing that $x_i - x_g = 0$, to say that the node eventually reaches the goal by stability. ■

**Invariant V.8.** Flocking

This is not possible in general here without a different control.

## VI. Future Work

## VII. Conclusions

## VIII. References

## References

[1] 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.