Multi-Agent Flocking Behavior using a Modified POMDP

Christopher Naughton (ID: cwnaught)

Abstract - In this project I attempt to model and simulate flocking behavior using a modified POMDP. I start with a description of the partially observable state-based model that I use to define the flocking problem. In it, each agent chooses an action to minimize its distance from a chosen leader-agent while avoiding collisions with the leader-agent and other flock members. Each agent must use a noisy observation of the other agents' locations to predict the next positions of both the target and the flock members. I solve the model using a modified search algorithm that uses the belief of each flock member's next positions to choose an action. I then ran and analyzed the simulation for a variety of flock sizes, observation noises, and domain sizes and compared the results, finding that the vague transition model favored algorithms that did not think too many steps ahead.

1. Introduction

Animal flocking behavior has long fascinated scientists and engineers alike. The emergent patterns from the collective motion of birds, fish, and insects are both beautiful to watch and difficult to understand. One of the first models of flocking behavior was Reynolds' "Boids" model, which consists of a set of universal rules that, when followed by each member of the flock, produce collective motion [1]. Researchers have extended these rules and built dynamic control models to help multi-robot systems exhibit this "flocking behavior" [2]. Unfortunately, these models either do not guarantee flocking behavior for large systems with random initializations or require complex dynamic models of all agents involved. This project represents flocking as a "target tracking" problem in which each member of the flock continually updates a belief state that describes the probable locations of the "leader" agent as well as all other flock members. Target tracking is a problem that has successfully been solved with approximate POMDP methods in the past [3].

2. Model Description

My simulation is a 2D discretized grid-world where each agent occupies one square of the grid. In total, there are N follower agents, referred to as "flock members", moving towards 1 leader agent, referred to as the "target". The state is defined by the 2D position and velocity of each agent. There are 9 actions available to each flock member: They can independently either increase, decrease, or maintain their velocity in the x and y directions. Since each agent can only change its velocity by 1, each flock member has a lot of inertia. This adds to the difficulty of control and should favor algorithms that look farther ahead. A flock member's action deterministically changes its own state. Each time step, every flock member incurs a penalty due to its speed, Manhattan distance from the target, and whether it collided with another agent. The Manhattan distance is the sum of the absolute values of the x and y distances. The cost formula is:

$$cost(B_i) = 5 * Dist(B_i, T) + 10 * Vel(B_i) + 50 * \#Collisions$$

where B_i represents flock member i and T represents the target. This cost is calculated for every flock member at each time step.

The leader agent, or "target", follows a predetermined trajectory independent of other flock members. This represents an agent whose actions are unpredictable, such as a human operator. Each "follower" agent sees a state consisting of its location, its velocity, and noisy sensor readings of both the target and any other flock members. A flock of N followers would therefore represent each state as a vector with N+2 dimensions. A factored graph of the POMDP system is shown in Figure 1 below. Known values, such as flock member position and velocity, are shaded in. Each follower sees itself as the agent and solves a separate POMDP based on the locations of the other flock members and the target.

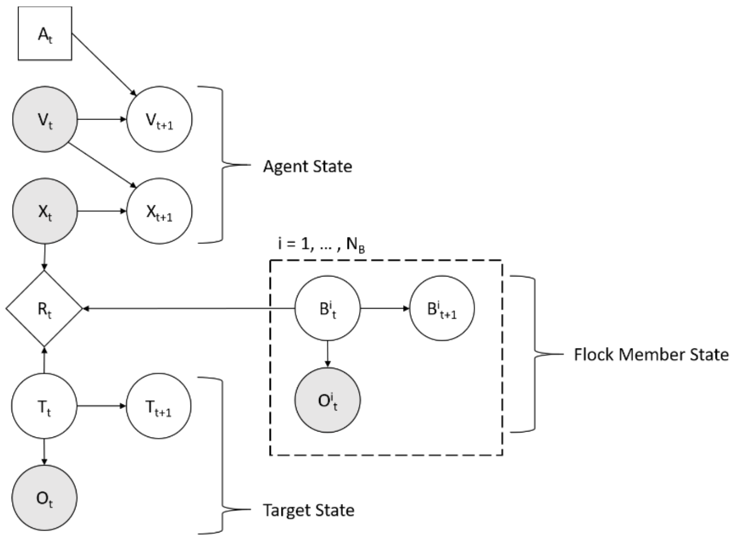

Figure 1: Factored POMDP representation of the Flocking Problem

3. Methods

The model is not framed well for classical POMDP solvers because each flock member's next position is a deterministic result of its action and state. Instead, the uncertainty is present in the flock member's awareness of the other agents' current and future positions. Figure 2 below summaries the algorithm.

```
Flocking Algorithm
Require: tEnd, N
initialize t = 0, cost = 0, pastBelief = [ ]
state ← randomFlock (N)
while t < tEnd:
    chosenActions = [ ]
    for each flock member i = 1, …, N:
        obs ← observe (state, flock member)
        belief ← updateBelief (obs, i, pastBelief [i])
        pastBelief [i] = belief
        mincost = 0
        for action in actionspace:
            cost ← lookAhead(state, belief, horizon)
            if cost < mincost:
                bestAction = action
        chosenActions[i] = bestAction
    state ← updateState (state, chosenActions)
    cost ← trueCost (state)
    t = t+1
```

Figure 2: Flocking Algorithm overview

In my solver each flock member receives a noisy observation of the current positions of every agent. The flock member's belief state is then updated based on the observation and the past belief state. This belief state is then used to recursively look ahead a specified number of time steps, known as "depth", and determine the action that gives the lowest expected cost. This minimum-cost action is saved, and once every agent has chosen an action the program updates the state and records the cost. The time counter increments and everything repeats for the specified number of time steps.

3.1 Observations

To simulate noisy sensor, each observed location is normally distributed about the real location. I control the noisiness of the observation using the standard deviation of the normal distribution. For low noise observations, I use a deviation of 0.1 spaces in each direction, and for high noise observations I use a deviation of 3 spaces in each direction. To reduce the computational time for large domains and flock sizes, I limit the flock member's field of view to a limited number of spaces in x and y. Flock members can't observe units outside of their individual fields of view. Figure 3 below shows the actual state, and an example of a low noise and high noise observation that the flock member receives. The heatmap represents the probability that the flock member believes the target is in each square. Note that the low noise observation is essentially deterministic, while the high noise observation is a cloud that isn't even centered on the actual target location.

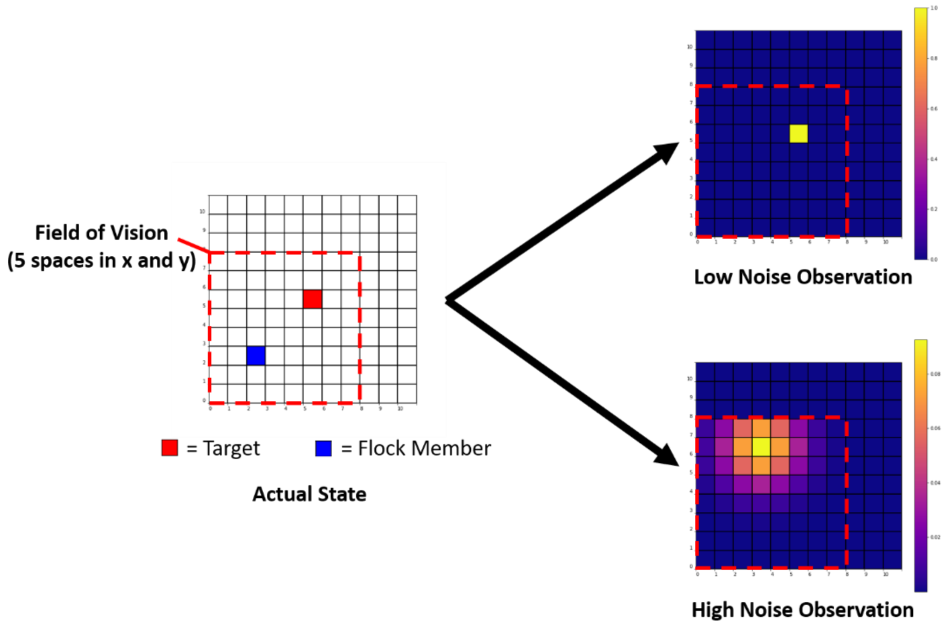

Figure 3: Example of low and high noise observations for a flock member

3.2 Belief states

For a flock of N follower-agents, each flock member has N+1 belief-vectors, one for each member and one for the target. The belief vector has length equal to the size of the visible domain, with each element representing a location in that domain. The vector element contains the probability that an agent is in that domain, with each location initialized as being uniformly probable.

I treat the belief state as a Hidden Markov Model, and so first update the current belief state using the probabilities from the observation. After iterating through every location in the visible domain and multiplying by the observation probabilities, I normalize over each visible location. This results in the belief state of the current location, and mathematically is represented as:

$$p\left(B_t^i \mid O_1^i, \dots, O_t^i\right) \propto p\left(B_t^i \mid O_1^i, \dots, O_{t-1}^i\right) * p\left(O_t^i \mid B_t^i\right)$$

Then, I estimate the next location of each flock member using a transition probability. To prevent the model from being dependent on the target dynamics I simply assume that each flock member will continue their current trajectory to find the next location, then use a normal distribution about that point. I iterate over each location in the visible domain, multiplying this normally distributed probability by the probability that the agent is in that location, and then normalize over the domain to get the updated belief state.

3.3 Lookahead

Lookahead calls itself recursively to a desired depth (or horizon), testing out each action and updating the belief state using the transition probability described in the previous section. Since the number of possible states is very large, I use beam search with a beam size of 5. In other words, each flock member only looks at the five most probable transitions for each other flock member. The cost for each step is calculated as:

$$totalCost = \sum_{s \text{ in possible states}} belief(s) * cost(s)$$

Unfortunately, for any simulation with a moving target this algorithm had a very large run time and poor performance for depths over 2 due to the transition probability's assumption that each agent will continue their course. As a result, all analysis in the next section will only focus on running this algorithm with a horizon of 1 or 2.

4. Results

To test this algorithm, I began with a single flock member moving over 20 time steps, and then varied 3 parameters: How the target moves, the noise in the observation, and the number of steps that each flock member thinks ahead (the horizon). For consistency, all six tests were performed in an 11x11 domain with the target either staying in the center ("stationary") or moving in a circle around the center ("circle"), and with the first flock member starting with the position x=2, y=2. Each flock member's field of vision was set to 10 tiles in each direction, so it could always see the entire domain. Table 1 below shows the average results from running each test 5 times. See Appendix A for a visual representation.

Model	Target Movement	Observation Deviation	Average Collisions	Average Cost	Average Run Time
Horizon = 1	Stationary	0.1 tiles	5	810	2.8 s
	Circle	0.1 tiles	0	1540	2.9 s
	Circle	3 tiles	0	1730	2.8 s
Horizon = 2	Stationary	0.1 tiles	1	536	17.7 s
	Circle	0.1 tiles	0	1407	17.5 s
	Circle	3 tiles	0	1809	17.5 s

Table 1: Algorithm performance as a function of target movement and observation uncertainty

As would be expected, the two-step horizon model performs better than the one-step horizon model in most cases, and the higher noise observation further increases the cost. One surprising result was that the two-step horizon performed worse than the one-step horizon when uncertainty was introduced into the moving target simulation. As was mentioned in the previous section, this is a result of the vague transition probability scheme. Since the lookahead assumes that each agent continues their previous trajectory, any uncertainty in these positions will confuse the two-step horizon more than the one-step horizon.

To further test the simulation, I looked at the effect of flock size on the average score per flock member. Figure 4 shows the results from a 21x21 grid-world domain tested over 40 time steps with the target circling the center and random initializations for each flock member. The field of vision remained was set at 10 squares from the flock member. Each flock-size simulation was run 5 times, and the error bars represent the 95% confidence interval.

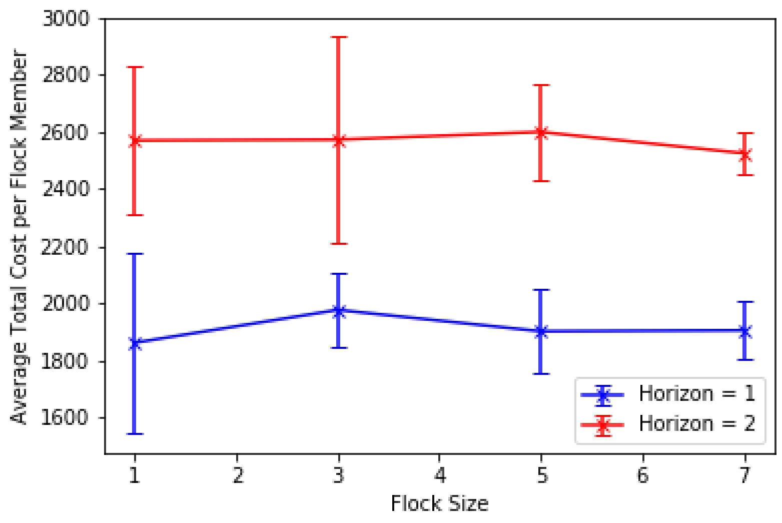

Figure 4: Single-step horizon outperforms two-step horizon for multiple flock sizes

Both algorithms seem invariant to flock size, with the confidence interval shrinking as the flock size increases. This time, however, the one-step horizon performs noticeably better than the two-step horizon. This is once again due to the transition probabilities not adequately describing the target motion, and the resulting errors propagating further for the two-step horizon than the one-step. The effect is more pronounced now because of the larger domain size and longer test duration. Please see Appendix A for still frames of the flock motion.

5. Conclusion and Future Work

In conclusion, flocking behavior can be modeled as a modified POMDP in which the uncertainty comes from noisy position measurements and predictions of future agent locations. This basic algorithm performs well in guiding flock members to follow the target while preventing collisions but favors smaller horizons due to the vague transition model. The model also scaled well with the addition of further flock members, and was able to follow a moving target without any knowledge of the target's pre-planned dynamics. Future work would be developing a more intricate transition model to allow for larger horizons, as well as experimenting with different action-spaces (such as every flock member moving at constant velocity and controlling their angle of movement) and costs (such as a cost incurred for being near other flock members).

6. References

[1] C. Reynolds. *Flocks, Herds, and Schools: A Distributed Behavioral Model*. Computer Graphics, pages 25-34, 1987

[2] B. Lei, W. Li, and F. Zhang. *Stable Flocking Algorithm for Multi-Robot Systems Formation Control*. IEEE Congress on Evolutionary Computation, 2008

[3] D. Hsu, W.S. Lee, and N. Rong. A Point-Based POMDP Planner for Target Tracking. IEEE Int. Conf. on Robotics & Automation, 2008

Appendix A: Still Frames of Flock Motion

Figures 5 and 6 below show still frames of subsequent time steps for a single flock member following a stationary target using a one and two-step horizon respectively. Notice that although both are very similar, the two-step horizon is better able to control its velocity and can get to the target faster and remain next to it for longer, while the one-step horizon has more trouble controlling its inertia.

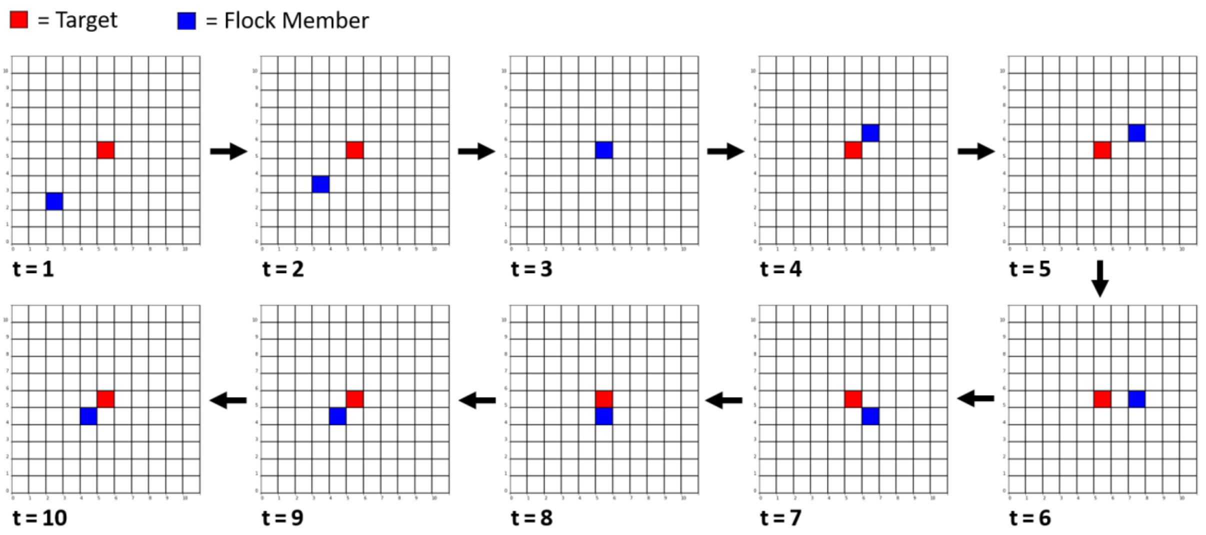

Figure 5: One-step horizon model with a stationary target and low noise observations

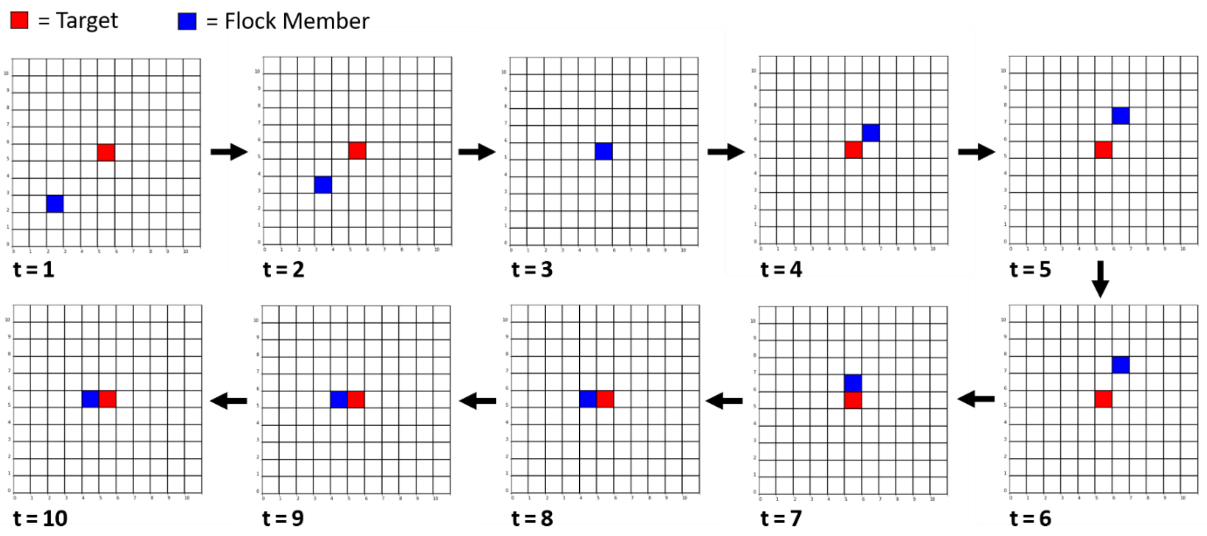

Figure 6: Two-step horizon model with a stationary target and low noise observations

Figure 7 on the next page shows flocking behavior for 7 flock members with a moving target and a one-step horizon model. Notice that since the field of vision is set at 10, the flock member in the bottom left corner moves randomly until the target comes into view. Also note that the flock members being to move less as they get closer to one another. A solution to this behavior might be to add a cost associated with being close to other flock members, thus encouraging all flock members to follow the target but leave enough distance between one another.

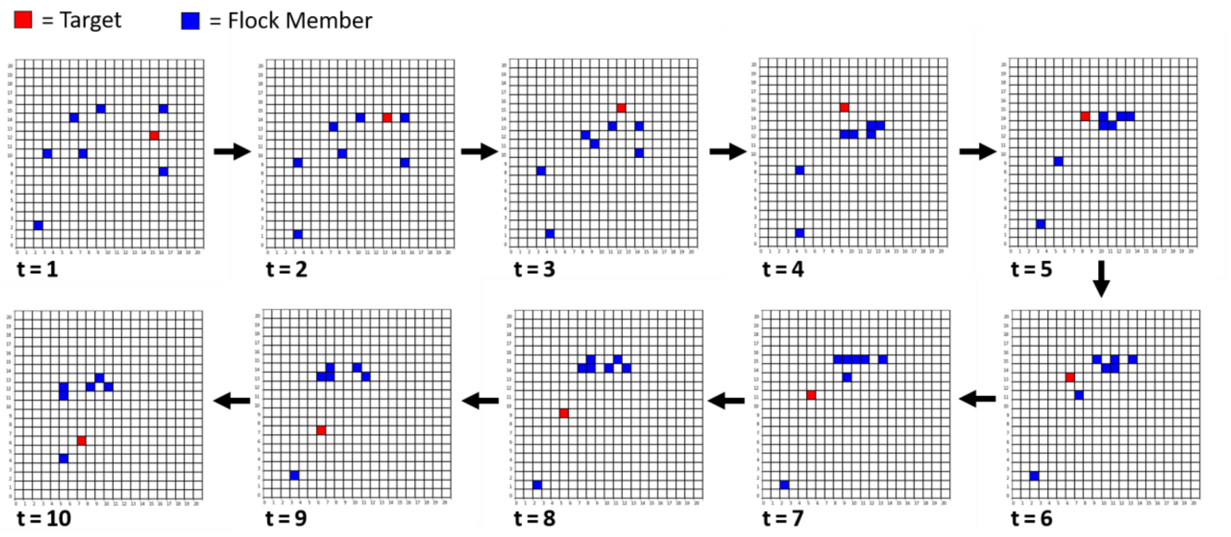

Figure 7: One-step horizon model with a moving target and low noise observations