# Agent-Based Modeling of Starling Murmurations

Abby Bratt

Joe Christianson

YP Kuo

December 12, 2016

## 1 Introduction

Many animals - insects, schooling fish, and flocking birds exhibit peculiar collective behavior. In particular, flocks of starlings (*Sturnus vulgaris*) are well-known for their rather spectacular flocking maneuvers, called murmurations. But while well-known, their swarm intelligence is not well-understood. It is hypothesized that they make these displays above their roost site in the evenings in order to ward off predators. There must be an underlying process that allows the birds to make rapid, drastic changes in density and shape while preserving the cohesion of the flock. However, this mechanism is still largely unknown.

Hildenbrandt et al. (2010) use an agent-based model (ABM) to test whether or not the highly complex patterns that we observe could be purely the result of self-organization. That is, they ask if the patterns we observe on a macroscopic level emerge purely from interactions between neighboring birds. Hildenbrandt et al. (2010) used their own proprietary model called StarDisplay. However, a problem with ABMs in general is that they are difficult to validate. For this model in particular, it is a logistical challenge to collect quantitative data on individual birds and flocks. So, Hildenbrandt et al. (2010) compare the results of their model qualitatively to video recordings and photographs of 10 large starling

flocks in Rome. They aim to replicate similar flock shapes and density gradients over time using mostly qualitative analyses. Ultimately, they were able to replicate many observed murmurations. Thus, they supported the hypothesis that such behavior is self-organizing.

In this paper, we implement the model described by Hildenbrandt et al. (2010) and test a new starling flight submodel. Without access to StarDisplay, we recreated their model and implemented our extension in MATLAB R2016b. However, in this process we noted a problem with the Hildenbrandt et al. (2010) paper. Some of the equations listed use ambiguous terms, do not produce the expected equilibria, or are not detailed in how to initialize in order to reproduce the same results. To combat this, we found an undergraduate thesis that did a similar analysis of the StarDisplay model and suggested improvements where it fails (van Opheusden, L.M.E, 2014).

Once we were able to reliably replicate model behavior under the same assumptions, we made some adjustments to the flight model. Hildenbrandt et al. (2010) use a simple fixed wing aerodynamics to get each bird's 3-dimensional flight force at each timestep. Clearly this is a simplifying assumption, as birds do not have fixed wings. According to Tobalske (2007), power is a u-shaped function of flight speed, and is influenced heavily by bird morphology. The morphology governs the intervals a bird must flap and glide to maintain pace and altitude. We used both a square wave and a sine wave to represent the force over a flapping starling wing in powered flight. This submodel is described in detail in the body of the text.

But importantly, this adjustment did not significantly change the behavior of the model. This is an important result as ABMs are notorious for their complexity and they rely on simplifying assumptions. The fact that the model was robust to changes confirms that those assumptions were valid in the first place.

# 2 Model Description

Put simply, this model is based on a number of forces. There is the flight force $F_{Flight}$ and the steering force, $F_{Steering}$, which is itself made up of four forces: a speed control force $f_\tau$, a roost attraction force $f_{Roost}$, a social force $f_{Social}$, and a random force $f_\zeta$. Here we describe each force in more detail, while Table 1 describes realistic parameters.

The model then uses Euler integration to calculate each bird's position and velocity, and banking angle in 3 dimensions at each time step. Key additional variables include the rotation and banking angles, which govern how individual birds turn, and the number of neighbors each individual has.

| Parameter | Description | Default value |
|---|---|---|
| $\Delta t$ | Integration time step | 5 ms |
| $\Delta u$ | Reaction time | 50 ms |
| $v_0$ | Cruise speed | 10 m/s |
| $M$ | Mass | 0.08 kg |
| $\frac{C_L}{C_D}$ | Lift-drag coefficient | 3.3 |
| $L_0$ | Default lift | 0.78 N |
| $D_0$ | Default thrust | 0.24 N |
| $T_0$ | Default thrust | 0.24 N |
| $w\beta_{in}$ | Banking control (in) | 10 |
| $w\beta_{out}$ | Banking control (out) | 1 |
| $\tau$ | Speed control relaxation time | 1 s |
| $R_{max}$ | Maximum perception radius | 100 m |
| $n_c$ | Topological range | 6 |
| $S$ | Interpolation factor | $0.005$ |
| $r_h$ | Radius of maximum separation (hard sphere) | 0.2 m |
| $r_{sep}$ | Separation radius | 4 m |
| $\Sigma$ | Parameter of the Gaussian $g(x)$ | 1.37 m |
| $w_s$ | Weighting factor separation force | 1 N |

| | | |
|---|---|---|
| $blindAngle$ | Rear "blind angle" cohesion and alignment | $90^o$ |
| $w_a$ | Weighting factor alignment force | 0.5 N |
| $w_c$ | Weighting factor cohesion force | 1 N |
| $C_c$ | Critical centrality below which an individual is assumed to be in the interior of a flock | 0.35 |
| $w_\zeta$ | Weighting factor random force | $0.10\overline{6}$ N |
| $R_{Roost}$ | Radius of roost | 150 m |
| $w_{RoostH}$ | Weighting factor horizontal attraction to roost | 0.1 N/m |
| $w_{RoostV}$ | Weighting factor vertical attraction to roost | 0.2 N |

Table 1: Default parameter values. Note that in the program we standardized all units in terms of kilograms, meters and seconds (Hildenbrandt et al., 2010)

## 2.1 Steering Force

### 2.1.1 Speed Control Force

The speed control force is part of the steering force. Starlings tend to fly at a constant cruising speed, $v_0$, but they may deviate to avoid collisions with their neighbors or to rejoin them (Hildenbrandt et al., 2010). So, we implement a speed control force which either accelerates or decelerates the bird to correct for deviations from $v_0$ according to the following equation:

$$\mathbf{f}_{\tau_i} = \frac{m}{\tau}(v_0 - v_i)\mathbf{e}_{x_i} \tag{1}$$

where, $v_i$ is the starling's velocity, and $\mathbf{e}_{x_i}$ its forward direction. Also recall from the table of parameters that $\tau$ is the relaxation time and $m$ is the mass of starling $i$ (Hildenbrandt et al., 2010).

### 2.1.2 Roost Attraction Force

Another component of the steering force is the roost attraction force. Recall the hypothesis that starlings perform these flocking maneuvers over the roost to ward of predators before nightfall. So, it makes sense that the flock tends to stay at roughly the same vertical and horizontal distance from the roost (in either direction - hence the $\pm$), subject to some deviations. The roost attraction force, like the speed control force, is what corrects for those deviations.

$$\mathbf{f}_{Roost_i} = \mathbf{f}_{RoostH_i} + \mathbf{f}_{RoostV_i} \tag{2}$$

where the horizontal roost attraction is given by

$$\mathbf{f}_{RoostH_i} = \pm w_{RoostH}\Big(\frac{1}{2} + \frac{1}{2}\Big(\mathbf{e}_{x_i}\mathbf{n}\Big)\Big)\mathbf{e}_{y_i} \tag{3}$$

and the vertical roost attraction is given by

$$\mathbf{f}_{RoostV_i} = -w_{RoostH}(p_{i_z})\mathbf{z} \tag{4}$$

where $\mathbf{z}$ is the vertical unit vector, $p_{i_z}$ is the vertical distance from the roost for bird $i$, and $\mathbf{e}_{x_i}$ and $\mathbf{e}_{y_i}$ are an individual's forward and vertical directions, respectively. Recall that $w_{RoostH}$ and $w_{RoostV}$ are weighting parameters taken from Table 1.

### 2.1.3 Social Force

The social force is the most complicated component of the model, but also the most important. It conveys the key assumption that interactions between birds occur on a purely local scale. Specifically, birds interact with others on a "topological range", $n_c$ which typically consists of their nearest 6 neighbors independent of their distance (Hildenbrandt et al., 2010). So, $n_c$ has a default value of 6.

In order to limit interactions to between nearest neighbors only, birds have an adaptive interaction range given by

$$R_i(t + \Delta u) = (1 - s)R_i(t) + s\Big(R_{max} - R_{max}\frac{|N_i(t)|}{n_c}\Big) \tag{5}$$

where $s$ is an interpolation factor, $R_{max}$ is the maxmimal metric interaction range and $N_i$, is the neighborhood of an individual given by the set of neighbors who's distance from bird $i$ is less than the adaptive interaction range, computed above. This makes intuitive sense since if the neighborhood is smaller than the optimal 6 neighbors, the interaction range will increase. If the neighborhood is larger, then the interaction range will decrease. This has the effect of keeping the number of influential neighbors a bird has more or less constant, with some variation when a flock splits or regroups, as an example.

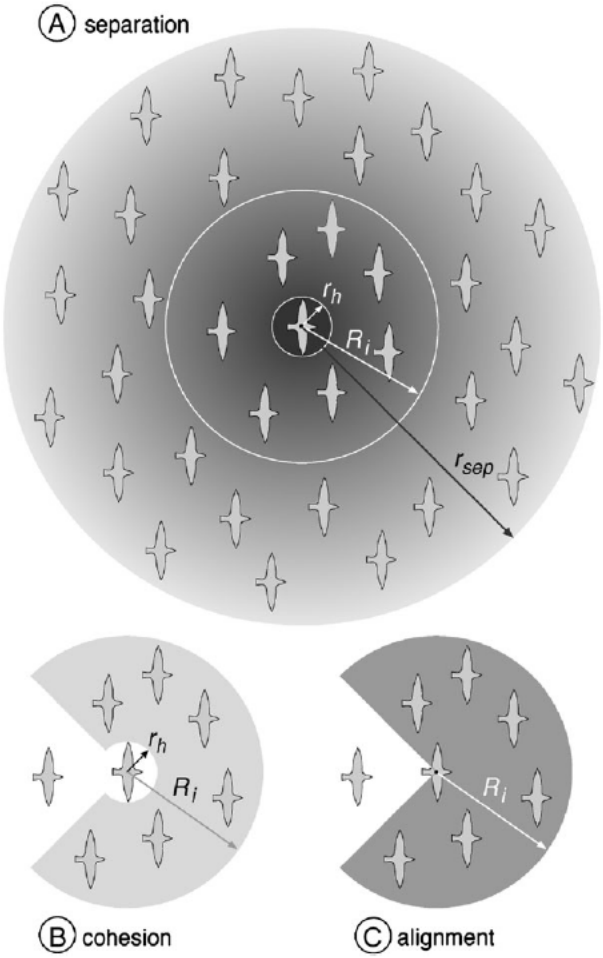


Figure 1: A) the radii influencing separation; B) the radii influencing cohesion; C) the radii influencing alignment

Once we have determined who each bird interacts with, we must determine how they

interact. This is described by 3 more forces: separation to avoid collision, and cohesion and alignment to maintain flock shape and density. A visual is given in 1.

The separation force pulls the bird away from the average position of the others in its neighborhood. Mathematically, this is given by

$$\mathbf{f}_{s_i} = -\frac{w_s}{|N_i(t)|} \sum_{i \in N_i(t)} g(d_{ij}) \mathbf{d}_{ij} \tag{6}$$

where $w_s$ is the separation weighting factor, $\mathbf{d}_{ij}$ is the direction from individual $i$ to inidividual $j$ specified by the unit vector and $g(d_{ij})$ is a halved Gaussian with mean $\mu$ and standard deviation $\sigma$ such that the separation force at the edge of the neighborhood is almost zero, keeping the birds somewhat close together. That is

$$g(d_{ij}) = \begin{cases} 1 & d_{ij} \leq r_h \\ exp\big( - \frac{(d_i j - r_h)^2}{\sigma^2} \big) & d_{ij} > r_h \end{cases}$$

where all variables follow from previous equations or are parameters that can be looked up in Table 1.

For cohesion, which attracts birds to the center of mass of their neighbors in opposition to the separation force. However, the two forces are not equal and opposite. Separation acts on a larger neighborhood than cohesion does. The reduced neighborhood used in computing the cohesion force includes all the birds in the full neighborhood, less those behind the bird. These are considered to be in the bird's blind angle, and therefore do not influence its behavior in this respect. They are included in the other force however, as it is still important to avoid collision with those behind you. Consider a person in traffic as an analogy. The cohesion force is given by

$$\mathbf{f}_{c_i} = C_i(t) \frac{w_c}{|N_i^*(t)|} \sum_{j \in N_i*(t)} X_{ij} \mathbf{d}_{ij} \tag{7}$$

where $w_c$ is the weighting factor for cohesion, $r_h$ is the radius of the hard sphere. $C_i$ is e measure of centrality - the length of the average direction vector pointing towards a bird's

neighbors.

$$g(X_{ij}) = \begin{cases} 0 & d_{ij} \leq r_h \\ 1 & d_{ij} > r_h \end{cases}$$

Finally, an alignment force acts on a bird, aligning it with the average forward direction of its neighbors in the same reduced neighborhood used in cohesion.

$$\mathbf{f}_{a_i} = \frac{w_a\Big(\sum_{j \in N_i^*} \mathbf{e}_{xj} - \mathbf{e}_{xi}\Big)}{||\sum_{j \in N_i^*} \mathbf{e}_{xj} - \mathbf{e}_{xi}||}\mathbf{d}_{ij} \tag{8}$$

### 2.1.4 Random Force

The random force is the final component of the steering force. It is there to impose relatively small stochastic influences on each individual birds' steering and contributes to the dramatic and rapid changes in overall flock shape and density that are characteristic of starlings. It is given by the following equation:

$$\mathbf{f}_{\zeta_i} = w_\zeta \cdot \boldsymbol{\zeta} \tag{9}$$

where $w_\zeta$ is the weighting factor from the random force described in Table 1, and $\boldsymbol{\zeta}$ is a random unit vector drawn from a uniform distribution.

The steering force itself is simply a sum of its four components. That is,

$$\mathbf{F}_{Steering_i} = \mathbf{f}_{Social_i} + \mathbf{f}_{\tau_i} + \mathbf{f}_{Roost_i} + \mathbf{f}_{\zeta_i} \tag{10}$$

## 2.2 Flight Force

### 2.2.1 Fixed Wing Aerodynamics

In their model, Hildenbrandt et al. (2010) used standard equations for fixed wing aerodynamics to simulate the flight of the birds. We used these same equations in early implementations in order to try and reproduce some of their figures. The equations for the lift, drag and thrust are given below:

$$L = \frac{1}{2}\rho S v^2 C_L \tag{11}$$

$$D = \frac{1}{2}\rho S v^2 C_D \tag{12}$$

Where $\rho$ is the air density and $S$ is the wing area of the bird. For the purposes of our model, we assume that these our constants. So, plugging in the default cruising speed $v_0$ into these equations yields $L_0$ and $D_0 = T_0$ respectively, the lift and the balanced drag and thrust at cruise speed, both of which are listed in Table 1. Then our simplified equations for lift and drag based on $L_0$ are:

$$L_i = \frac{v_i^2}{v_0^2} L_0 = \frac{v_i^2}{v_0^2} mg \tag{13}$$

$$D_i = \frac{C_D}{C_L} L_i = \frac{C_D v_i^2}{C_L v_0^2} mg \tag{14}$$

Thus, the flight forces are given by

$$\mathbf{F}_{Flight_i} = (\mathbf{L}_i + \mathbf{D}_i + \mathbf{T}_0 + \mathbf{L}_i + \mathbf{mg}) \tag{15}$$

where

$$\mathbf{L}_i = L_i \cdot \mathbf{e}_{z_i} \tag{16}$$

$$\mathbf{D}_i = -D_i \cdot \mathbf{e}_{x_i} \tag{17}$$

$$\mathbf{T}_i = T_0 \cdot \mathbf{e}_{x_i} \tag{18}$$

But when flying along a curve there is the additional complication of accounting roll, pitch and yaw. That is, the angles at which the bird is turning in each dimension, as shown in Figure 2

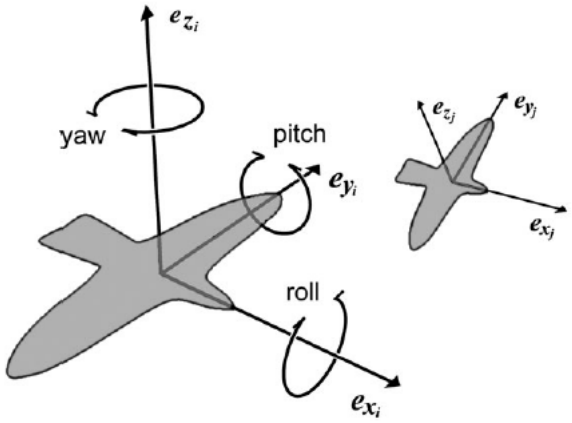


Figure 2: Rotation around the 3 axes, where $\mathbf{e}_{x,y,z}$ are bird $i$'s direction along each axis.

Birds roll into the direction of their turn, up to a variable banking angle. Intuitively, their tendency to roll into a turn increases with their tendency to turn sideward in coordination with their neighbors and to stay close to the roost.

We first compute the bird's lateral acceleration from the steering force.

$$\mathbf{a}_{l_i} = \left(\frac{\mathbf{F}_{Steering_i}}{}\right)\mathbf{e}_{y_i} \tag{19}$$

From the direction of this vector, we and compute the interior banking angle, and their exterior banking angle. These are given by

$$tan(\beta_{in_i}) = w_{\beta_{in}}||\mathbf{a}_{l_i}||\Delta t \tag{20}$$

$$tan(\beta_{out_i}) = w_{\beta_{out}} sin(\beta_i)\Delta t \tag{21}$$

Finally, we use Euler integraton over these forces to compute the velocity $v_i$, position $p_i$ and banking angle $\beta_i$ at each time step using

$$v_i(t+\Delta t) = v_i(t) + \frac{1}{m}(\mathbf{F}_{Steering_i}(t) + \mathbf{F}_{Flight_i}(t))\Delta t \tag{22}$$

$$p_i(t+\Delta t) = p_i(t) + v_i(t+\Delta t)\Delta t \tag{23}$$

$$\beta_i(t+\Delta t) = \beta_i(t) + \beta_{in_i} - \beta_{out_i} \tag{24}$$

### 2.2.2 Variable Thrust

**Biomechanics of Bird Flight** For birds a fixed wing flight model is obviously a simplifying assumption. A better model would incorporate the oscillatory thrust that is inherent to the flapping and gliding required to remain suspended. So, our extension of the Hildenbrandt et al. (2010) model was to implement a more realistic flight model.

Flapping flight and fixed wing flight are fundamentally different. In fixed wing flight thrust, is produced separately from lift. The geometry of the air frame and, more importantly, how it influences drag and lift coefficient remains constant in flight duration. This is obviously not true in bird flight, where wings not only generate lift but also thrust. The main point in this section is to factor this major distinction between bird flight mechanics and simple fixed wing aerodynamics in to the model, making it less "simplistic." It turns out this small update to the model is extremely challenging analytically and computationally. The difficulties arise from the need to recalculate coefficients for aerodynamic forces due to changing air frame as the bird flaps its wings. The equation to use is one of the fundamental equations in aerodynamics given below

$$F_{aero} = \oiint \vec{p} \cdot d\vec{A} \tag{25}$$

To summarize, for any object immersed in a fluid, the mechanical forces are transmitted at every point on the surface of the body $A$. The forces are transmitted through the pressure $\vec{p}$, which acts perpendicular to the surface $d\vec{A}$. The net force can be found by integrating the dot product of pressure and the surface normal around the entire surface. For a moving flow, the pressure will vary from point to point because the velocity varies from point to point. For some simple flow problems, we can determine the pressure distribution (and the net force) if we know the velocity distribution by using Bernoulli's equation in compressible fluid, given by

$$\frac{v^2}{2} + gz + \frac{p}{\rho} = constant, \tag{26}$$

where $v$ is the fluid flow speed at a point on a streamline, $g$ is the acceleration due to

gravity, $z$ is the elevation of the point above a reference plane, with the positive z-direction pointing upward, $\rho$, is the density of the fluid at all points in the fluid. For a long time in aircraft design industries, more complex flow problems have been studied using wind tunnel testing to determine the aerodynamic force instead of computer simulations. Most of these calculations were done on fixed wing aircrafts where the surface does not change over time. The difficulties that arise from performing said calculation are (1) finding a formula representation of the manifold of the bird's surface, and (2) updating it as time progresses.

The point being made is that we do not have the resources to make any attempts on detailed modeling of bird flight mechanics fruitful. An oversimplification has to be made in order to make any significant progress. Therefore, we decided to treat the overall geometry of the bird unchanged throughout the flight duration and only vary the amount of thrust that is generated depending on the stage of wing oscillation. This seemed like a great compromise, but it allows us to continue using the aerodynamic coefficients provided in the paper without doing any complex calculations.

Simplistic models of how thrust and lift is produced in birds look at momentum transfer between the bird and air surrounding it. A bird uses its wings to generate lift by accelerating masses of air downwards and thrust by accelerating air backwards. Rough estimates of mechanical energy involved in the thrust and lift forces at uniform speeds can be obtained by looking at the vertical and horizontal forces separately (Videler, 2005). For generation of lift we look at the affected air mass that is accelerated by the beating wings. Approximate the cross section of incoming air mass affected by the wings motion as a circle with circumference $b$. The linear density of the air mass:

$$\frac{dm}{dx} = \frac{dm}{dV}\frac{dV}{dx} = \rho A = \frac{1}{2}\rho\pi\Big(\frac{b}{2}\Big)^2 \tag{27}$$

If the bird is moving with horizontal velocity of $v$, then the incoming air velocity experi-

enced by the bird is $-v$. The mass flux of air is:

$$\frac{dm}{dt} = \frac{dm}{dx}\frac{dx}{dt} = -\frac{1}{2}\rho v\pi\left(\frac{b}{2}\right)^2 \tag{28}$$

With the wings in stationary state, the vertical velocity of the air is zero since the bird is assumed to have no vertical velocity. As the bird flaps its wings downward, momentum is transfered to the incoming air mass. Let $w$ be the final vertical speed of the air, then the momentum per unit time is

$$\frac{dp}{dt} = -v\frac{dm}{dt} \tag{29}$$

By conservation of linear momentum, downward momentum of air mass results in upward momentum of the bird. Since we still assume the bird has no vertical velocity after the momentum exchange, the change in momentum per unit time must be equal to the gravitational force on the bird. The power (energy per unit time) transferred to the incoming air is given by:

$$P = \frac{dE}{dt} = \frac{d}{dt}\left(\frac{p^2}{2m}\right) = \frac{1}{2}\frac{M^2g^2}{\pi(b/2)^2\rho v} \tag{30}$$

The power required to maintain the momentum transfer to the air is:

$$P_i = \frac{1}{2}\frac{m^2g^2}{\pi(b/2)^2\rho v} \tag{31}$$

Calculating the power transfer to the incoming air as thrust is simple. If the bird maintains its current horizontal velocity, the thrust power has to equal to the power lost to atmospheric drag. The power required to balance the drag forces is

$$P_d = F_d v = D_0\frac{v^3}{v_0^2} \tag{32}$$

Therefore, the total power loss is given by

$$P = P_i + P_d = \frac{1}{2}\frac{m^2g^2}{\pi(b/2)^2\rho} + D_0\frac{v^3}{v_0^2} \tag{33}$$

Plotting the total power loss with respect to velocity yields what is called the flight power curve, shown below.

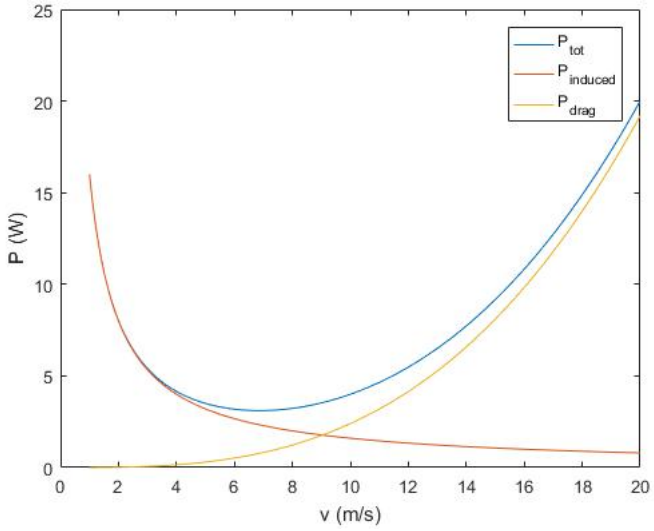


Figure 3: Power curve plotted using Matlab

**Implementation** Since movement of the wings during flight is the most energy intensive locomotion in all of vertibrates, the U-shaped curve is a valuable tool for comparing different flight patterns or behavioral strategies (Tobalske, 2007). Two main points of interest on U-shaped curve are the minimum power speed $v_{mp}$ and maximum range speed $v_{mr}$. These characteristic speeds represent one of the most obvious ways in which the biomechanics of flight may be integrated with behavioral ecology. Often, ecological studies of flight would begin by assuming that birds select $v_{mp}$ for aerial foraging or searching and select $v_{mr}$ for long-distance flight such as migration (Tobalske, 2007).

An obvious next step to make in making the model more realistic would be to calculate $v_{mr}$ and $v_{mp}$ for starling. This is exactly what we did, and the results were problematic. The calculated values were $v_{mp} = 6\ m/s$ and $v_{mp} = 8.1\ m/s$, nowhere near the cursing speed of $10\ m/s$ the paper proposed. Our options were (1) to use the calculated $v_{mp}$ or (2) to continue using the proposed cruising speed of $10\ m/s$ and abandon the entire idea of a power curve. However, we were quick to realize that the assumptions we made in deriving the equations for power loss were so simplistic that the only thing true about it is the general shape of the curve. Important factors that could contribute to the low $v_{mp}$ and $v_{mr}$ were (1)

ignoring the vortex produced by air of different speed mixing and (2) ignoring the lift due to geometry of the bird in calculation of downward airspeed. A corrected formula for finding $v_{mp}$ (Pennycuick, 2001)

$$v_{mp} = 0.807 \frac{k^{0.25} m^{0.5} g^{0.5}}{\rho^{0.5} b^{0.5} C_D^{0.25}}. \tag{34}$$

gave a value $v_{mp} = 10.2$ m/s. This agrees with the cruising speed Hildenbrandt et al. (2010) used in the model. Pennycuick (2001) also gave an equation for calculating wing beat frequency:

$$f = m^{3/8} g^{1/2} b^{23/24} S^{1/3} \rho^{3/8}, \tag{35}$$

which gave $f = 10$ Hz.

With these two pieces of valuable information we constructed two periodic functions that represents the oscillatory motion of wing flapping. The two functions are sinusoidal and square, with amplitude of the thrust calculated to maintain a constant speed of 10 m/s. The mathematical descriptions are given below:

$$f_{sin}(t) = D_0 \big[ \sin(2\pi f t) - 1 \big] \tag{36}$$

$$f_{square}(t) = D_0 \big[ \text{square}(2\pi f t) - 1 \big] \tag{37}$$

# 3 Reproduction of Results

In addition to the implementation complexity, we faced other challenges as well. Our implementation ended up being computationally intensive. If $n$ is the number of birds and $t$ is the number of time steps, then our simulation run time scales like $tn^2$. The chief bottleneck is that each timestep, a distance matrix between all the birds must be computed. For reference, simulating 200 birds over 24,000 timesteps (2 minutes), took just under 90 minutes to run on an i7 processor. The problem is somewhat parallelizable, because the computation of each bird only relies on data from the previous timestep. Matlab, however, locks their multicore programming ability in a toolbox. We converted the code to NumPy,

but the single threaded implementation was around twice as slow, which made any gain from parallelizing dubious especially with developer time taken into account.

Another challenge was that the reference paper (Hildenbrandt et al., 2010) did not publish their reference dataset of the real starlings gathered in Rome.

Our replication effort had two main approaches. First, we were able to replicate several of the figures from (van Opheusden, L.M.E, 2014), that showed individual bird behaviors in isolation. Second, we were able to validate that complex behavior is produced by birds that maintain a fairly constant number of interaction partners. Figures 4, 5, and 6 refer to van Opheusden's work, and are direct recreations of several of his figures showing individual bird behaviors under specific forces.

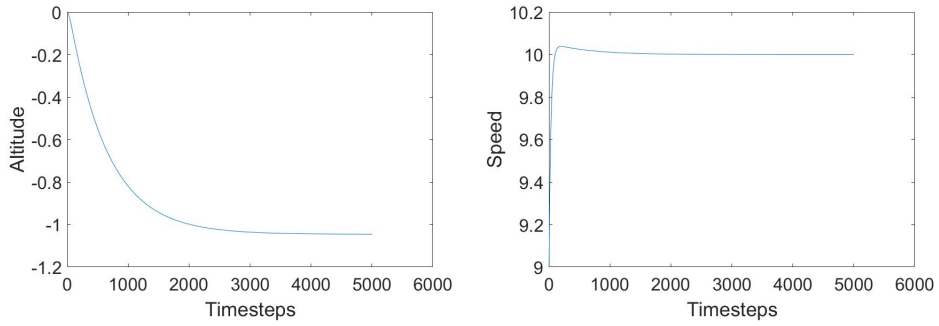


Figure 4: This mirrors figure 2.4 in van Opheusden, L.M.E, 2014. It depicts a bird that starts at the origin with an initial velocity vector of $\mathbf{v} = (v_0 - 1, 0, 0)^T$ where $v_0 = 10m/s$

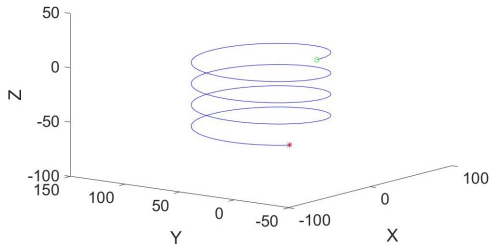


Figure 5: This mirrors figure 2.6 in van Opheusden, L.M.E, 2014. It depicts a bird that starts at $\mathbf{p} = (0, -1, 10)^T$ with an initial velocity of $\mathbf{v} = (v0, 0, 0)^10$ and a constant fixed banking angle of $\beta = 0.2$. As expected, it spirals downward.

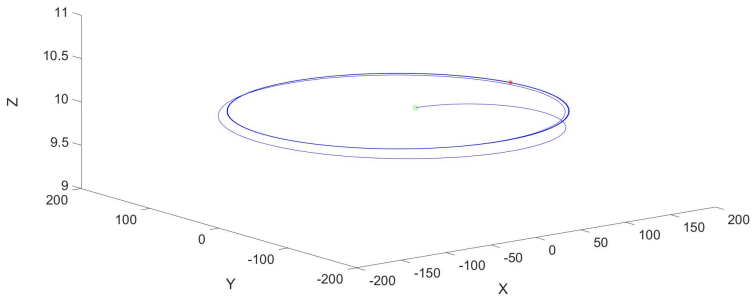


Figure 6: This mirrors figure 2.9 in van Opheusden, L.M.E, 2014. It depicts a bird that starts at $\mathbf{p} = (20, 1, 10)^T$ with initial velocity $\mathbf{v} = (10, 0, 0)^T$, as expected converges to the radius of the roost.

In terms of replicating the original paper (Hildenbrandt et al., 2010) as stated before, we were challenged because we did not have the original dataset of starling behavior, nor did we have the computing power to simulate a model on the same scale as them. We were able to run many simulations of flocks of a few dozen birds, and a few of a few hundred. These flock sizes were sufficient to observe the birds self organizing into groups. These groups were larger than their desired interaction partners, supporting the idea that the model can simulate complex behavior of groups of birds without each bird being connected to every other bird.

We can think of the flock behavior as being transmitted through the flock as each bird aligns with those around it, like traffic. In our case, because we could not simulate enough birds, the density was too low for that behavior to be transmitted globally throughout the entire flock. Because experimentally it the behavior was shared between birds that were not immediate neighbors, so we feel confident saying that a more a larger run of our simulation would have reflected the desired effects. Below are three figures quantifying this conclusion of successful self-organization.

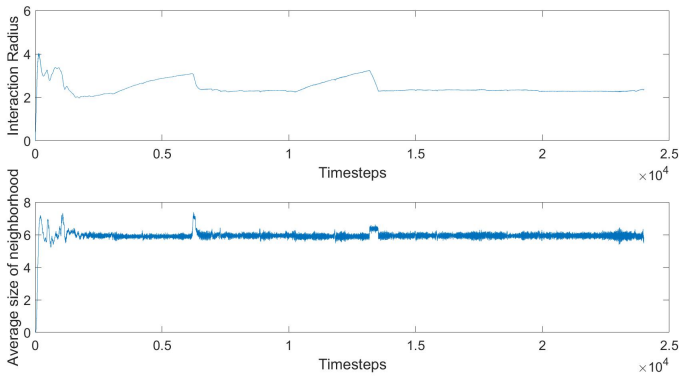


Figure 7: This graph confirms the chief behavior that the original StarDisplay model sought to show. The up graph shows how large a sphere each birds searches on average to find neighbors, and the bottom shows the average number of neighbors each bird bases it's action off of each timestep. This data is from a simulation of 200 birds over 24,000 timesteps.

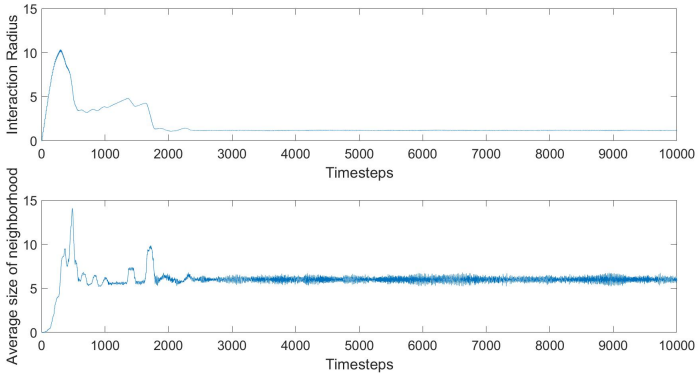


Figure 8: This graph shows the same thing as fig. 7. It is gathered from a simulation of 30 birds over 10,000 timesteps.

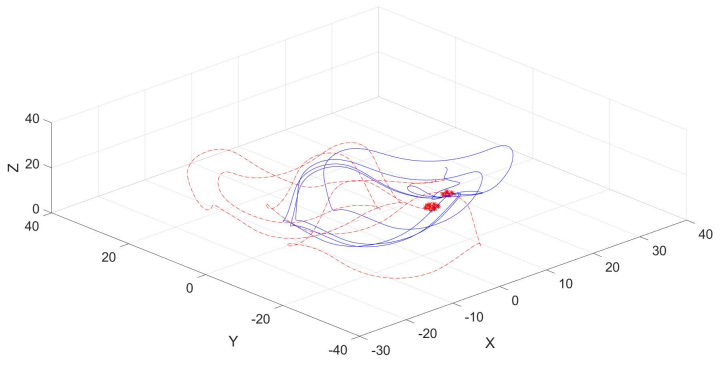


Figure 9: This graph shows the flight paths of the two emergent groups in the simulation from fig. 8. There are 30 birds total, that self organize into 2 groups, which implies one group will have at least 15 birds. We can see from the the graph that the average number of interaction partners holds steady at 6, which means these flocks are successfully organizing without each bird interacting with all the birds in its group.

In terms of our implementation, there are a few details we should call attention to.

Based on van Opheusden's derivation of the roost attraction force steady state, we know that the radius of orbit is:

$$r = \frac{2mv_0^2}{w_{Roost}}. \tag{38}$$

With the given values and the prescribed weighting factor, $w_{Roost}$ of 0.01, the radius of orbit ends up being 1600 meters which is far too large. Van Opheuden's recommendation is to increase the value of $w_{Roost}$ until $r = R_{Roost} = 150$. We do so, resulting in a weighting factor of $w_{Roost} = 0.10\overline{6}$.

In many places we divide by the size of a particular neighborhood. Sometimes, particularly early on, birds will not have found any neighbors yet, and these will result in divide by zero errors. We create a function called `safe_length`, which returns 0.9 as the value for the length if it is zero. Not too much consideration was given to the choice of 0.9 because in all cases where we are dividing by the length, we are averaging over a set of values for the neighborhood. The neighborhood being empty will cause these values to be 0 anyway,

and so the `safe_length` constant is merely to prevent converting zero vectors to `NaN` vectors.

Van Opheuden is critical of the original update equation, because it causes the bird to spin under normal turning conditions, and is not grounded in any physical idea of angular momentum. We also noted this issue. Van Opheuden's solution is to introduce an angular velocity state to each bird, compute an angular acceleration, and so update banking using that information. Our solution was simpler, but effective at reducing problematic states due to spin. We simply clamp the banking angle to be in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

# 4 Novel Results

Coming up with the mathematical description of variable thrust analytically was a long process, but implementing it to generate results was relatively easy. After replacing the constant thrust model with the variable thrust model, we found out most behaviors of StarDisplay did not change drastically. This was not to our surprise, since the wingbeat frequency of starlings is about 10 Hz. Solving numerically for horizontal velocity in ODE involving atmospheric drag with variable driving force only produces microscopic deviations from the constant thrust model.

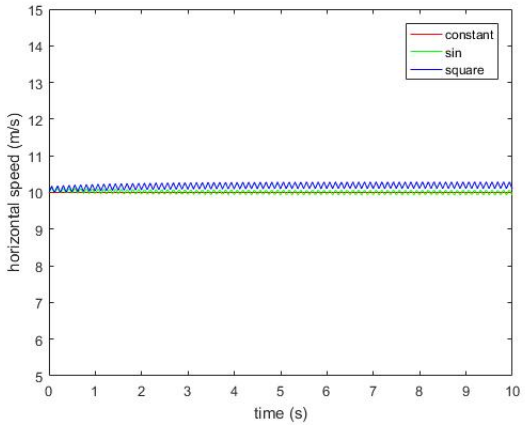


Figure 10: This graph shows the horizontal velocity under different thrust model. The variable thrusts preserve the same macroscopic behavior and only add in small osiliations in velocity.

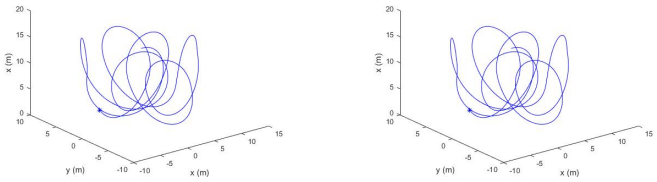

Figure 11: Attitude control test with constant thrust (left) and sinusoidal thrust (right). The two plots suggest the two models produced the same macroscopic behaviors.

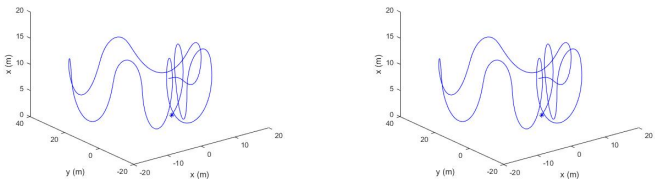

Figure 12: Banking test with constant thrust (left) and sinusoidal thrust (right). The two plots showed striking similarities.

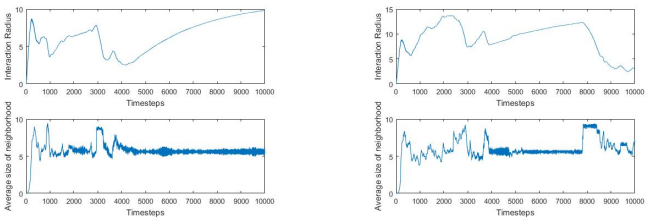


Figure 13: Interaction range(up) and average neighborhood(down) with sinusoidal thrust (left) and square thrust (right) for 30 birds and 10000 timsteps. The two plots showed average neighborhood size is in the range of 5-10 birds agreeing with the constant thrust model.

# 5 Conclusions

We were able to successfully replicate many of the starling murmuration behavior described in literature and in StarDisplay, further supporting the assumption that these complex patterns arise from local interactions between a small number of nearest neighbors. This is an important result as ABMs whose individuals exhibit simple cognition are generally preferable to their more complex counterparts. In particular, when we added complexity to the model, in the form of the more biologically appropriate variable thrust flight model, we did not find that it significantly changed the behavior of the model. Therefore, in future implementations of the model, it would be fine to make the same assumption as Hildenbrandt et al. (2010) do and use a simpler fixed flight model.

Further work would include correcting some of the limitations with the StarDisplay model though, many of which are pointed out by van Opheusden, L.M.E (2014) and ourselves. Additionally we would like to improve the performance of the model in Matlab. We attempted to do this in Matlab, as described above without much success. However, if we vectorized our functions in Matlab, we may see significant upticks in the model runtime. This would allow us to test much larger flock sizes than we were able to in this paper. This is important since starling flocks in the wild can be up to 10,000 birds, so it is crucial to test

whether or not the model holds for flocks of this size. If it does, it could prove a useful tool for ornithologists since the empirical study of huge swarms is often costly, labor intensive, and logistically challenging.

# References


Hildenbrandt, H., Carere, C., and Hemelrijk, C. (2010). Self-organized aerial displays of thousands of starlings: a model. *Behavioral Ecology*, 21(6):1349–1359.

Pennycuick, C. J. (2001). Speeds and wingbeat frequencies of migrating birds compared with calculated benchmarks. *The Journal of Experimental Biology*, 204:3283–3294.

Tedrake, R., Jackowski, Z., Cory, R., Roberts, J. W., and Hoburg, W. (2009). *Learning to fly like a bird*. PhD thesis, Massachusetts Institute of Technology: Computer Science and Artificial Intelligence Lab.

Tobalske, B. W. (2007). Biomechanics of bird flight. *The Journal of Experimental Biology*, 210:3135–3146.

van Opheusden, L.M.E (2014). *Birds of a feather flock together: an analysis of starling swarm intelligence in the StarDisplay model*. PhD thesis, University of Twente.

Videler, J. J. (2005). *Avian Flight*. Oxford University Press.

Wu, J.-c. and Popovic, Z. (2003). Realistic modeling of bird flight animations. *ACM Transactions on Graphics*.

# A Appendices

## A.1 Implementation of the model in Matlab

This function is responsible for converting from the forward direction ($\mathbf{e_x}$, given by normalizing the velocity) and the roll angle $\beta$, and returning the other two orientation vectors $\mathbf{e_y}$ and $\mathbf{e_z}$.

```
function [ey, ez] = fwdDirAndBeta2basis(fwdDir, beta)
    % Assume the fwdDir is normalized!
    normxy = norm(fwdDir(1:2));
    cosyaw = fwdDir(1) / normxy;
    sinyaw = fwdDir(2) / normxy;
    if isnan(cosyaw) || isnan(sinyaw)
        % assumes birds are never vertical 'gimbal lock'
        cosyaw = 0;
        sinyaw = sign(fwdDir(2));
    end

    cospitch = norm(fwdDir(1:2));
    sinpitch = fwdDir(3);
    if isnan(cospitch) || isnan(sinpitch)
        cospitch = sign(fwdDir(1));
        sinpitch = 0;
    end
    cosroll = cos(beta);
    sinroll = sin(beta);

    ey = [sinroll*sinpitch*cosyaw - sinyaw*cosroll;
        sinroll*sinpitch*sinyaw + cosyaw*cosroll;
        -sinroll*cospitch];
    ez = [-cosroll*sinpitch*cosyaw - sinroll*sinyaw;
```

```
        -cosroll*sinpitch*sinyaw + cosyaw*sinroll;
        cosroll*cospitch];
end
```

As described above, this function simply helps eliminate divide by zero issues when averaging values over a neighborhood of birds.

```
function [l] = safe_length(arr)
    l = length(arr);
    if ~l
        l = 0.9;
    end
end
```

Next we have our load constants script, which helps keep the variables we want to think of as constants separate from the rest of the code.

```
dt = 0.005;
v0 = 10;
mass = 0.08;
gravity = 9.81;
CD_CL = 3.3;
L0 = mass * gravity;
T0 = CD_CL * L0;
Tau = 1;
z0 = 10;
Rroost = 150;
du = 0.05;
s = 0.005;
Rmax = 100;
nc = 6;
rh = 0.2;
```

```
sigma = 1.77;
rsep = 4;

liftConstant = L0/v0^2;
dragConstant = -CD_CL/v0^2 * mass * gravity;
gravityForce = -[0;0;mass*gravity];

wAlt = 0.2;
wRoost = 0.1066666; %Adjusted from 0.01
wSigma = 0.01;
ws = 1;
wc = 1;
wa = 0.5;
wBin = 1;
wBout = 5;

wr = s/du;
```

Next we have the a script to initialize the constants of a simulation. We kept it as a separate file to make switching between different initial conditions easier. It is aptly called `setInitialConditions.m`.

```
NumBirds = 30;
NumTimeSteps = 5000;
posOverTime = zeros(3, NumBirds, NumTimeSteps);
velOverTime = zeros(3, NumBirds, NumTimeSteps);
bankingOverTime = zeros(NumBirds, NumTimeSteps);
interactionRadiusOverTime = zeros(NumBirds, NumTimeSteps);
posOverTime(:, 1:NumBirds, 1) = [20*(2*rand(2, NumBirds, 1) - ones(2, NumBirds)); ...
    5*rand(1, NumBirds) - 5 + z0];
velOverTime(:, 1:NumBirds, 1) = [v0*rand(2, NumBirds); zeros(1, NumBirds)];
```

Finally we have our main script, `Simulation.m`, which iterates over the time steps,

and each step iterates over all the birds.

```
close all; clear all; clc;

loadConstants;
setInitialConditions;

% deltaVecs(:, itr, jtr) is the vector from bird itr to bird jtr.
deltaVecs = zeros(3, NumBirds, NumBirds);
% deltaDists(:, itr, jtr) is the distance between bird itr and bird jtr.
deltaDists = zeros(NumBirds, NumBirds);
speeds=zeros(NumBirds, 1);

%% - Simulation
tic;
for timeStep=1:NumTimeSteps
    for itr=1:NumBirds
        for jtr=1:NumBirds
            deltaVecs(:, itr, jtr) = posOverTime(:, jtr, timeStep) - ...
                posOverTime(:, itr, timeStep);
            deltaDists(itr, jtr) = norm(deltaVecs(:, itr, jtr));
        end
    end
    speeds = sqrt(sum(velOverTime(:, :, timeStep).^2));
    forwardDirections = velOverTime(:, :, timeStep) ./ speeds;
    for bird=1:NumBirds
        % Bird Properties \/==================
        fwdDir = forwardDirections(:, bird);
        [wingDir, upDir] = fwdDirAndBeta2basis(fwdDir, ...
            bankingOverTime(bird, timeStep));
        speed = speeds(bird);
        position = posOverTime(:, bird, timeStep);
```

```
interRadius = interactionRadiusOverTime(bird, timeStep);
neighbors = [1:NumBirds] .* double(...
    deltaDists(bird,:) < interRadius);
neighbors = neighbors(neighbors~=0);
neighbors = neighbors(neighbors~=bird);
angles = acos(squeeze(sum(...
    deltaVecs(:, bird, neighbors).*fwdDir))'./squeeze(...
    deltaDists(bird, neighbors)));
inView = double(angles < 3*pi/4) + double(angles > 5*pi/4);
inViewNeighbors = neighbors(inView ~= 0);
reducedNeighbors = [1:NumBirds] .* double(deltaDists(bird, :)...
     < (2 * interRadius));
reducedNeighbors = reducedNeighbors(reducedNeighbors ~= 0);
reducedNeighbors = reducedNeighbors(reducedNeighbors ~= bird);

% Bird Properites /\==================
% Interaction Forces \/===============
g = exp(-(deltaDists(bird,neighbors) - rh).^2 / sigma^2)...
    .*double(deltaDists(bird,neighbors) > rh) ...
    + double(deltaDists(bird,neighbors) <= rh);
separationForce = -ws/safe_length(neighbors) * sum(g .* ...
    squeeze(deltaVecs(:,bird,neighbors))./repmat(...
    deltaDists(bird, neighbors), 3, 1), 2);

centrality = 1/(safe_length(reducedNeighbors)) * norm(sum(...
    squeeze(deltaVecs(:, bird, reducedNeighbors)), 2));
thresholdDistances = double(deltaDists(bird, inViewNeighbors)...
    > rh);
cohesion = centrality * ...
    wc/safe_length(inViewNeighbors) * squeeze(...
    deltaVecs(:, bird, inViewNeighbors))*thresholdDistances';
```

```
diffInDirections = sum(forwardDirections(:, inViewNeighbors)...
    - fwdDir, 2);
if norm(diffInDirections) > 0
    alignmentForce = wa * diffInDirections / ...
        norm(diffInDirections);
else
    alignmentForce = [0;0;0];
end
% Interaction Forces /\===============
% SteeringForces \/===================
speedControlForce = mass/Tau * (v0 - speed) * fwdDir;
altitudeControlForce = -wAlt * (position(Z) - z0) * [0;0;1];

% Attraction To Roost
normalToRoost = [position(1:2);0];
normalToRoost = normalToRoost/norm(normalToRoost);
roostAttractionForce = -sign(dot(normalToRoost, wingDir)) * ...
    wRoost * (.5 + .5 * dot(fwdDir, normalToRoost)) * wingDir;

% Random Force
randVec = (2 * [rand; rand; rand] - [1;1;1]);
randomForce = wSigma * randVec / norm(randVec);

steeringForce = speedControlForce + altitudeControlForce + ...
    roostAttractionForce + randomForce + separationForce + ...
    cohesion + alignmentForce;

% SteeringForces /\===================
% FlightForces \/====================
liftForce = liftConstant * speed^2* upDir;
dragForce = dragConstant * speed^2 * fwdDir;
thrustForce = T0 * fwdDir;
% gravityForce never changes and so is precomputed in the loading
```

```
        % constants phase.
        flightForce = liftForce + dragForce + thrustForce + gravityForce;
        % FlightForces /\====================
        force = steeringForce + flightForce;

        % Banking Angle Equations
        bankingIn = atan(wBin * dot(steeringForce, wingDir) / mass * dt);
        bankingOut = atan(wBout*sin(bankingOverTime(bird, timeStep))*dt);
        % update equations
        velOverTime(:, bird, timeStep + 1) = ...
            velOverTime(:, bird, timeStep) + force/mass * dt;
        posOverTime(:, bird, timeStep + 1) = ...
            posOverTime(:, bird, timeStep) + ...
            velOverTime(:, bird, timeStep + 1) * dt;
        interactionRadiusOverTime(bird, timeStep + (du/dt)) = max(...
            interRadius + wr * (1 - safe_length(neighbors)/nc) * ...
            (Rmax - interRadius) * du, 0);
        bankingOverTime(bird, timeStep + 1) = median([...
            bankingOverTime(bird, timeStep) + bankingIn - bankingOut;
            -pi/2;
            pi/2]);
    end
    toc;
    if mod(timeStep, 100) == 0
        fprintf('Completed Timestep %d', timeStep);
    end
end
```

## A.2 Implementation of our extension in Matlab

I used the following code to generate the flight power curve and calculating $v_{mp}$ and $v_{mr}$

```
% generates flight power curve
v0 = 10;
M = 0.08;
g = 9.81;
L0 = 0.78;
D0 = 0.24;
b=0.20;
rho = 1.225;

v=1:0.1:20;

pi = (M*g)^2./(pi*(b/2)^2*rho.*v);
pd = D0/v0^2*v.^3;

p = pd+pi;

% plotting flight power curve
figure
plot(v,p)
hold on
plot(v,pi)
plot(v,pd)
hold off
legend('P_{tot}','P_{induced}','P_{drag}')
xlabel('v (m/s)');
ylabel('P (W)')

% find vmp
vmp = find(p==min(p))*0.1

% find vmr
vmr = find(p./v==min(p./v))*0.1
```

I used the following code to generate the variable thrust

```
% Produces periodic thrust
% mode 1 = sinusoidal
% mode 2 = square wave
% wing beat frequency calculated form PREDICTING WINGBEAT
% FREQUENCY AND WAVELENGTH OF BIRDS BY C. J. PENNYCUICK

function flapthrust = flapping(T,mode)
    flapBeatFrequency = 8;
    typethrust =  [sin(2*pi*(flapBeatFrequency)*T)+1, ...
        (square(2*pi*(flapBeatFrequency)*T)+1)];
    flapthrust = typethrust(mode);
end
```

The following code sovles ODE involving drag and different models of thrust

```
% Testing variable thrust
dt = 0.005;
v0 = 10;
g = 9.81;
M = 0.08;
L0 = 0.78;
D0 = 0.24;
T0 = D0;
Tmax = 10;

t = 0:dt:Tmax;

v = zeros(2,length(t));

v(:,1) = 10;
```

```
for ti=1:length(t)-1
    v(1,ti+1) = (-D0*(v(1,ti)/v0)^2+T0*flapping(ti*dt,1))*dt/M+v(1,ti);
    v(2,ti+1) = (-D0*(v(2,ti)/v0)^2+T0*flapping(ti*dt,2))*dt/M+v(2,ti);
end

figure
plot(t,10*ones(1,length(t)),'r');
hold on
plot(t,v(1,:),'g');
plot(t,v(2,:),'b');
hold off

xlabel('time (s)');
ylabel('horizontal speed (m/s)')
legend('constant','sin','square')
axis([0 inf 5 15 ])
```

## A.3 Visualization of model output in C

To create timestep by timestep visualizations of the flocking behavior, we modified a differential equation grapher written in C called Euler3. Euler3 takes in a text file structured as follows, where the subscripts indicate the identity of the bird and the position dimension.

| | | | | | |
|---|---|---|---|---|---|
| Timestep 1 | $position_{1x}$ | $position_{1y}$ | $position_{1z}$ | $position_{2x}$ | ... |
| Timestep 2 | $position_{1x}$ | $position_{1y}$ | $position_{1z}$ | $position_{2x}$ | ... |
| ... | | | | | |

The actual code can be found at: https://github.com/alephic/euler3/blob/master/euler3.c