Project II: Big Brother is Watching You!

Albert Quizon, Christopher Silvia, Siyu Yang

2015-11-02

1 Instructions

2 Model Assumptions

3 Parameter Values and their Justifications

3.1 Gotham City

As the original construction in the Batman Series, Gotham City is New York City Mahattan area below 14th Street. For simplicity, we are assuming that Gotham City is a 3 miles by 3 miles square area. Gotham University (a fictional depiction of NYU) and the Financial District are reduced respectively to a 0.5×0.5 and a 1×1 square miles square areas. Assume the two areas are seperate.

3.2 Technical Data for UAV

insert other stuff Speed of UAV is considered 16 meters per second.

3.3 Ground Sampling Distance

The work *Drones And Aerial Observation* introduced an equation regarding Ground Sampling Distance (GSD) the area a drone will be able to observe at a given cruise height:

$$GSD = \frac{\text{pixel size x height above ground level}}{\text{focal length}} \tag{1}$$

If we assume that the drone is armed with Canon S100, a most common filming device used by UAVs, we will have a camera with pixel size of 0.0019mm, focal length ranging from 24 120mm, and can produce images of up to 4,000 by 4,000 pixels. Assuming a cruise height of 1000 meters, we will be able to get a 1000 ft by 1000 ft surveillance vision every given moment.

4 Simulation Algorithm

4.1 Model I: Rectangular Patrol Path

For the first simple discrete model, we are going to estimate where every single drone would cover a given square area on the map. For any given drone, it will be patrolling a given rectangular area. We need to make sure in the interval of 15 minutes (900 seconds), the drone will cover the whole area. Thus we are considering a 304.8 meter (1000 feet) wide "snake" drawing out a rectangular region for all 15 minutes, with ground speed 16 meters per second (35.71 miles per hour). This would cover:

$$35.71^{\text{mi}}/_{\text{hr}} \times 15minute \times 1000feet = 1.7miles^2 \tag{2}$$

To cover all of Gotham City (assumed as a 4828 by 4828 meters^2 square = 23,309,584 $[m^2]$) in 15 minutes, we need at least 6 drones ($\frac{23309584}{4389120} = 5.31$).

In order to increase patroling around Gotham University and Financial district to every 5 minutes, and the rest of the area every 20 minutes, we have the designated patrol area of every drone around critical region:

$$35.71^{\text{mi}}/_{\text{hr}} \times 5minute \times 1000feet = 0.6miles^2 \tag{3}$$

and patrol area of every drone around none-critical region:

$$35.71^{\text{mi}}/_{\text{hr}} \times 20minute \times 1000feet = 2.5miles^2 \tag{4}$$

Which means we need 1 drone to continuously patrol the 0.25 square miles of region around Gotham University, and 2 drones to patrol the area around Financial district. For the rest of area, we would need 3 other drones to cover. We can easily implement the need for refueling every 5 hours by providing another set of 6 drones which will take over for the first set of drones while they recharge. We make the assumption that drones can recharge/refuel in under 5 hours. This means the city needs at least 12 drones to monitor that no point in the city goes unobserved for more than 15 minutes in a row.

We can see that the above model is very crude and limited. It assumes that the drones would draw out $9miles \times 1000feet$ rectangles on the map for 15 minutes, without considering refueling, turns and cost. Areas around Gotham University and Financial districts are over patrolled, resulting in higher cost. Moreover, the patrol routes are fixed, resulting in no randomness and thus could easily calculate a jaywalk route without getting caught. Based on the weaknesses above, we present the second model.

4.2 Model II: Ideal Gas Model

The second model we present is a Ideal Gas Model were drones fly on random directions in uniform speed. When a drone reaches the boundary of the surveillance region, that drone"bounces off" (see Figure 1). Note that if the wall is vertical, the bouncing just multiplies the x component of the velocity by -1.

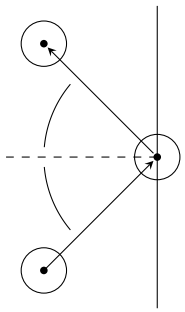

Figure 1: An incoming drone bounces off of a barrier. The circle is its radius of vision.

If the drones do not interact with each other, and are given uniformly random velocities and positions, then they should be distributed uniformly throughout the surveillence domain.

We have implemented a simulation of this model in matlab. A typical moment in time looks like this:

Notice that in this case, the drones' fields of view are overlapping.

5 Code Appendix

Listing 1: Ideal Gas Simulation

```
% model parameters
w = 2.0; % width of container
h = 3.0; % height of container
r = 0.1; % radius of drone vision

eps = 1e-13;
huge = 1e15;

% simulation parameters
n_steps = 100000; % number of events
n_drones = 100; % number of drones at a time

% set up randomly distributed velocites all with
%       unit magnitude
v = zeros(n_drones,2);
for i = 1:n_drones
        theta = 2*pi*rand();
        v(i,:) = [cos(theta), sin(theta)];
end

% place drones randomly within the domain,
%       but a distance of no less than r away from
```

```
%       the walls
x = [r, r] + [w-r,h-r] .* rand(n_drones,2);

% array to save the drones in
X = zeros(n_drones, n_steps, 2);

t = 0;
time_intervals = zeros(1,n_steps); % save event times

for i = 1:n_steps
    % for each drone, find the time until it collides with each wall
        tls = (r - x(:,1))./v(:,1);
        trs = (w - r - x(:,1))./v(:,1);
        tbs = (r - x(:,2))./v(:,2);
        tts = (h - r - x(:,2))./v(:,2);

        % for each wall, find the minimum positive time until a collision
        %       with it, and which drone this time corresponds to.
        [tl, il] = min(tls + (huge*(tls < eps)));
        [tr, ir] = min(trs + (huge*(trs < eps)));
        [tb, ib] = min(tbs + (huge*(tbs < eps)));
        [tt, it] = min(tts + (huge*(tts < eps)));

        % find the minimum overall time until a wall collision,
        %       and which wall will be collided with
    [t_so_far, which_wall] = min([tl, tr, tb, tt]);

        % move the drones forwards and advance time
```

```
        x += v * t_so_far;
        t += t_so_far;
        printf('.')

        % if the drone collides with the left or right wall, invert the x veloci
        % if it collides with the top or bottom wall, invert the y velocity
        if which_wall == 1
                v(il, 1) = -v(il,1);
        end
        if which_wall == 2
                v(ir, 1) = -v(ir,1);
        end
        if which_wall == 3
                v(ib, 2) = -v(ib,2);
        end
        if which_wall == 4
                v(it, 2) = -v(it,2);
        end

        % record the timestamp and the drone positions at this time
        time_intervals(i) = t_so_far;
        X(:,i,:) = x;
end
```