Project II: Big Brother is Watching You!

Albert Quizon, Christopher Silvia, Siyu Yang

2015-11-02

Abstract

abstract-text

1 Instructions

The problem requires us to develop a MAV surveillance scheme across a hypothetical area – Gotham City, with the goal to completely patrol the given area. The scheme are subject to the four following constraints:

- Every given geographic point in the city needs to be covered on 15 minutes periods
- The flight plan needs to miniminze sharp turns, take into consideration refueling stops, and a possible scheme of 30
- The flight plan needs to cover the area around Gotham University and Financial Districts on 5 minutes periods, and Gotham Central Park once in 20 minutes
- The flight plan needs to be randomized for the sake of fairness against ”insider jaywalking”

All four constraints need not to be satisfied at the same time.

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), the Financial District and Gotham Central Park are reduced respectively to a 0.5×0.5, 1×1 square miles square areas and a 1×1 square miles square areas. Assume the two areas are seperate.

3.2 Technical Data for UAV

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.

From *Drones And Aerial Observation*, we know that for common UAV used in mapping and survellance, it is reasonable to consider its flight speed as 16 meters per second.

3.3 Assumptions for drone behaviors

We will make the following

- Drones can start anywhere and can come down for recharging/refueling at any point on the grid.

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). The path covered would be individualized for each drone, as in Figure 1. This would cover:

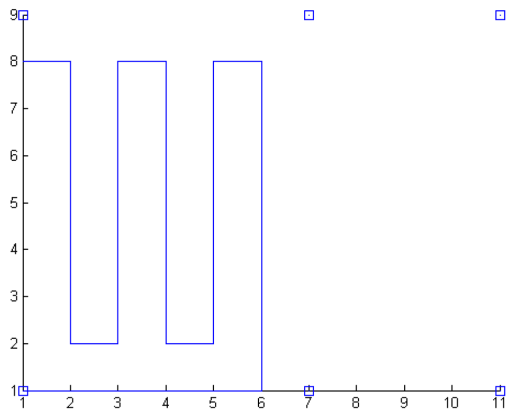


Figure 1: Path of one drone after one cycle of 15 minutes.

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

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 ($\frac{23309584}{4389120} = 5.31$).

We can see that the above model is very crude and limited. It assumes that the drones would draw out 4,389,120 [m^2] rectangles on the map for 15 minutes (without considering refueling).

In order to create a model for this discrete method, we had to assume a square map of Gotham, where a drone can observe a square area of 304.8 meters by 304.8 meters. We had to break down the Gotham area into a discrete set of grid points, where the spacing between each point was twice the radius of observation of a drone (304.8 meters). After doing some calculations shown in the code provided in Appendix Listing 2, we determined we needed 256 "blocks" to cover all of Gotham. Each drone could cover 48 of those blocks in a 15 minute interval. We used two arrays. One array was designed to keep track of the current location of each drone and another array was designed to keep track of time since that area was last observed. We do have to make the assumption that the drones can start anywhere and can come down for recharging/refueling at any point on the grid. We implemented a check to see if any block in the array was not observed in a 15 minute interval. A visual of the paths and positions the drones took are provided in Figure 2.

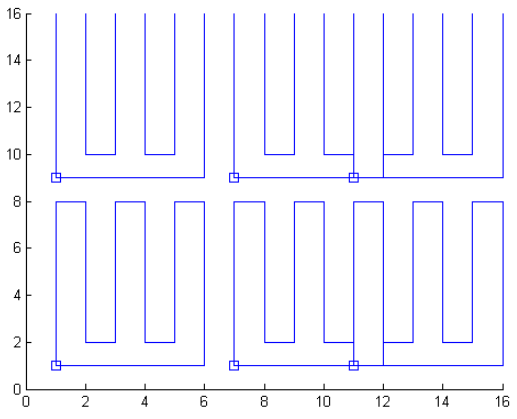


Figure 2: Path of 6 drones after one cycle of 15 minutes.

The blue squares represent the location of each drone at a given time, and the blue lines are the path that each drone took. There were no failures to observe any point in the grid after running the code for multiple cycles. There were some overlaps in the paths of some drones, so some areas of the map were observed more frequently than once every 15 minutes but the figure verifies that we need at least 6 drones watching the city at all times.

The better alternative to this form of solution is to create a large U-based path that traverses the whole array. This allows for even spacing between the drones, so all the areas of the map will be observed at the same frequency. While this method provides a better distribution of observation time to each point, it also safeguards against any failures in drones. If a percentage of the drones fail, then we will still maintain coverage of the whole area of Gotham without having to reprogram any drones or send new ones. The drones would take on a path similar to the one shown in Figure 3.

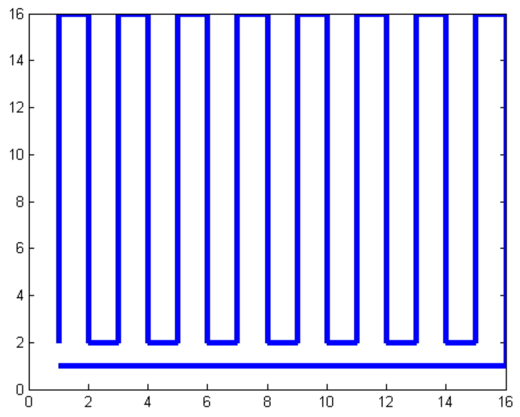


Figure 3: Path of 6 drones in a large comb pattern.

4.2 Strategy II: Variable Levels of Coverage

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, we implete a discrete strategy similar to the one offered above. In this case, to increase the patrol period means to decrease the area patroled:

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

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

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

With simple math we can find that this 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 including Gotham Central Park, 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 also very 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.

4.3 Strategy III: Contingencies

This model involves the usage of fuel with respect to the motion of the drone. Paths requiring more sharp turns will require more frequent recharging/refueling stops. We will also have to consider that the MAVs are not very reliable so a significant proportion of them might be briefly grounded for repairs. However, we have to implement a plan to ensure that even if some drones go down, all areas of the city will be observed even if somewhat less often. We make the assumption that 30 % of the drones will become unusable.

The first problem to address is the use of fuel. More turns means more fuel consumed, so the number of terms is our constraint. We can still assume that each drone can observe their own area, so we have to apply a minimization of the number of turns for the path of each drone. We can also assume that the areas still have to be observed at least once every 15 minutes. There are a few conditions regarding the path of each drone:

1. The path of the drone must touch each node of its respective grid.
2. The path of the drone cannot self-intersect.
3. The path of the drone must return to start at the end of 15 minutes.

We must satisfy all of these conditions. There are only two general paths we can draw to fulfill these conditions. The first is an H-like path, as shown in Figure 4a. The second is a U-like path, as shown in Figure 4b. The H path needs 11 turns to complete the path while the U path only needs 7 turns. Therefore, the most fuel efficient path is the U path.

There is another problem of having the drones refuel at least every 5 hours. The optimal number of drones in this case would be based on the amount of time needed for refueling, which we assume to be less than the length of time it can fly for. Based on this assumption, we have a formula to determine how many drones we need. O is original number of drones needed, N is number of backup drones needed (drone stations where we can recharge), L is length of duration of flight for each drone, R is time needed to refuel. The first assumption we need in this case is to stagger when we put each drone up in the air. The maximum time we can stagger each drone by is:

$$T_{stagger} = \frac{L}{O}N$$

This is dependent on how many backup drones we have (which we can treat as drone stations to simplify things). Therefore, each drone that comes into each station will be empty on fuel and will need to completely refuel. Our constraint then is that the time interval has to be greater than the refueling time.

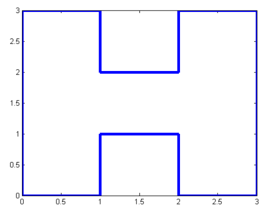
$$\begin{aligned} \frac{L}{O}N &\geq R \\ N &\geq \frac{(R)(O)}{T} \end{aligned}$$

As a result, Number of drones $\geq \frac{(Time\ needed\ to\ refuel)(Original\ Number\ of\ Drones)}{(Length\ of\ duration\ of\ flight)}$.

We have O = 6 drones, and L = 5 hours, but now we can solve for number of drones given the time needed to refuel. If the drones need 5 hours to completely refuel, then we would need 12 drones.

The next problem is to consider contingency plans for when some drones become unusable.

If 30 % of the drones become unusable, then the simplest way of addressing the issue is to just replace them by having 30 % more drones to replace the ones that break. We will need at least 12 drones to patrol every point at least once every 15 minutes. From this, we know we will need a total of 18 drones, where 30 % of these (5.4) will become unusable. When any drone fails, we can just send another up in its place. There is also the option of reprogramming the drones to assume new paths. This is the most basic strategy to this problem.


(a) H Path of a drone given even number of grid squares

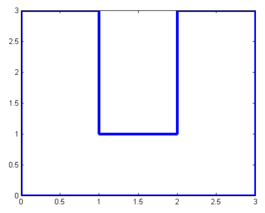

(b) U Path of a drone given even number of grid squares

Figure 4: H and U Paths of a drone

However, our goal should be to implement a strategy which covers all of the area even if 30 % of the drones go down. Ideally, we should not have to reprogram the drones or send more in from storage. The way to do this is by creating a large cycle which should minimize the number of turns and fuel. We already proved earlier that the best shape for the paths in a given area are U shaped, so we can implement a ”comb” like path to traverse the whole map, like in Figure 5.

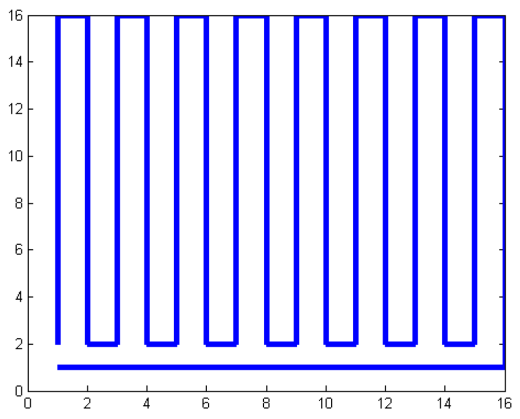


Figure 5: Path of 6 drones in a large comb pattern.

To go further with this model, we can actually create a comb like pattern for the entire grid and use the drones to traverse the comb pattern to accomplish surveillance. We will still have 256 grid points to observe, but if we evenly distribute the 6 drones along the path, we can then have them observe 43 (256/6, rounded up) spaces each before the next drone retraces those same spaces. The time it takes to traverse a path of 43 tiles is 806.25 seconds, or 13.44 minutes. From this, we can determine that each area in Gotham city under this method will be observed at least once every 13.44 minutes, which is even better than the desired 15 minutes.

Now we have to account for the 30 % rate that the drones will fail. That means that of the current 6 that we have running, about 2 of them will fail. We will then have a total of 256 tiles to cover between 4 drones.

The best case would be if the two drones were not consecutive and only left gaps of 86 tiles. This means that any point in the map could go unobserved for 1612.5 seconds, or 26.875 minutes.

In the worst case, the 2 drones that become unusable are consecutive drones along the path. Since the drones were split apart by 43 tiles, then we will be left with an 129 tile gap. This means that any point in the map can go unobserved for a maximum of 2418.75 seconds or 40.3125 minutes.

Given a strategy to assume a "comb" like path and a 30 % failure rate of drones, the maximum any point in the map can go unobserved for is 40.3125 minutes and the minimum is 27.875 minutes.

4.4 Strategy IV: Ideal Gas Model

4.4.1 Problem Statement

For the final scenario, we need to program the drone paths so that our insider knowledge about the drones' plans cannot be used to give us (the modelers) the ability to plan an observation-free jaywalking path through Gotham. This severely limits the amount of structure our model can have. Our model cannot have a fixed schedule for drone movements; we (the modelers) would then be able to avoid the drones. We could have random drones scheduled on fixed routes, but our knowledge of the routes would allow us (the modelers) to go to low-observability regions, and perhaps use spotters to determine when drones were dispatched on routes.

4.4.2 Description of Model

We have therefore chosen not to use fixed routes for this model. Instead, the drones will travel in straight lines at fixed velocities within the borders of Gotham, and bounce off of the boundaries when they reach them (See Figure 6). If the drones occasionally change their directions randomly, then they should be uniformly distributed throughout Gotham. If the drones are uniformly distributed throughout Gotham, then the modelers will not be able to plan low-observability paths, since each part of Gotham has the same chance of being observed as any other.

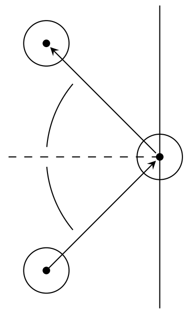


Figure 6: An incoming drone bounces off of a barrier. The circle is its radius of vision. Note that if the wall extends in the y direction, the bouncing just multiplies the x component of the velocity by -1.

Suppose there is a domain with area A, which contains a single drone. The drone moves at a constant velocity v, and observes a radius r around it. The direction of the drone, and its position, are both uniformly random at time

$t = 0$. Consider a target point, x_t, within A. We are interested in how likely the drone is to intersect the point x_t in a given time interval. This situation is shown in Figure **??**.

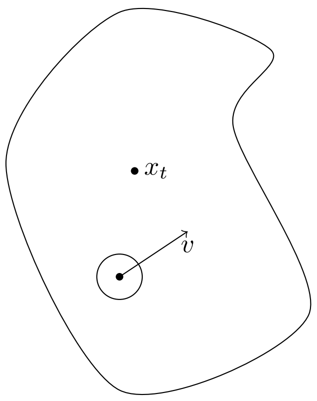


Figure 7: A drone moves through A with constant velocity v

4.4.3 Analytical Formula for Observation CDF

We are interested in determining whether the target is observed between times 0 and t. Since the drone is uniformly distributed throughout the domain A, as long as it is not near the boundary of A, the drone should fly in a straight path covering a distance vt, and observe the area shown below:

Since the drone's position and velocity are uniformly randomly distributed, the shape in Figure **??** is approximately equally likely to cover any point in A. Approximation errors come from the boundaries of the region: if the drone bounces off the boundary between $t = 0$ and $t = t$, then the area covered is not $\pi r^2 + 2r(vt)$. If there is significantly more area than boundary, however, than the chance of being observed between $t = 0$ and $t = t$ is just the fraction of A which is observed, which is:

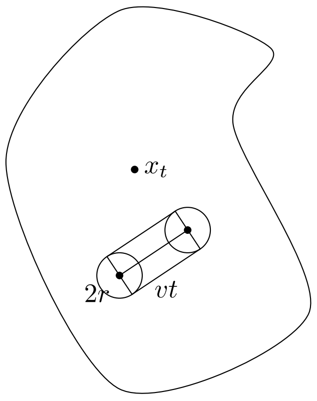


Figure 8: In time t, the drone covers area $\pi r^2 + 2r(vt)$

$$P(t) = \frac{\pi r^2 + 2r(vt)}{A} \tag{5}$$

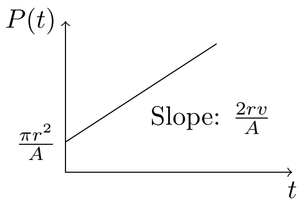


Figure 9: Plot of the Cumulative Probability of being observed by a single drone, between times 0 and t. Note that this model breaks down at or before $t = \frac{A-\pi r^2}{2rv}$, when the model predicts $P(t) = 1$.

There is a more rigerous derivation of this probability in Appendix ????.

4.4.4 Multiple Drones

If there are multiple drones, and the probability of being observed by one of them between $t = 0$ and $t = t_f$ is $P(t)$, then the probability of being observed by any one of them is equal to the probability of *not* being observed by *none* of them:

Between $t = 0$ and $t = t_f$, for t_f small	Probability
Observation from a single drone.	$P(t)$
No Observation from a single drone.	$1 - P(t)$
No Observation from n independant drones	$(1 - P(t))^n$
At least one Observation from n independant drones	$1 - (1 - P(t))^n$

Therefore, when $t << \frac{A - \pi r^2}{2rv}$, the probability of being observed by at least one drone in the interval from 0 to t is given by:

$$P_n(t) = 1 - \left(1 - \frac{\pi r^2 + 2r(vt)}{A}\right)^n \tag{6}$$

Notice that for any λ, $P_n(t)$ is unchanged if the parameters r, v, and A are rescaled as follows:

$$r \to \lambda r$$
$$v \to \lambda v$$
$$A \to \lambda^2 A$$

To see this, we can apply this rescaling to $P_n(t)$:

$$\begin{aligned}
1 - \left(1 - \frac{\pi r^2 + 2r(vt)}{A}\right)^n &\to 1 - \left(1 - \frac{\pi(\lambda r)^2 + 2(\lambda r)((\lambda v)t)}{A}\right)^n \\
&\to 1 - \left(1 - \frac{\lambda^2}{\lambda^2}\frac{\pi r^2 + 2r(vt)}{A}\right)^n \\
P_n(t) &\to P_n(t)
\end{aligned}$$

Therefore, $P_n(t)$ is independant of the "length-scale" of the problem. If we express time in units of $\frac{1}{v}$, then the value of v becomes unimportant. We can chose $\lambda = +\frac{1}{\sqrt{A}}$, so that the rescaled A is 1. Therefore, the only relevant quantity is the normalized radius of the drone, which is best expressed by the ratio $\frac{r}{A}$.

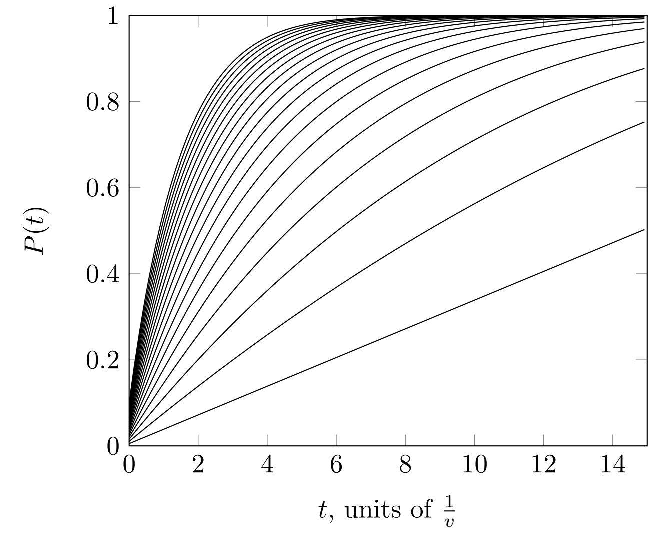
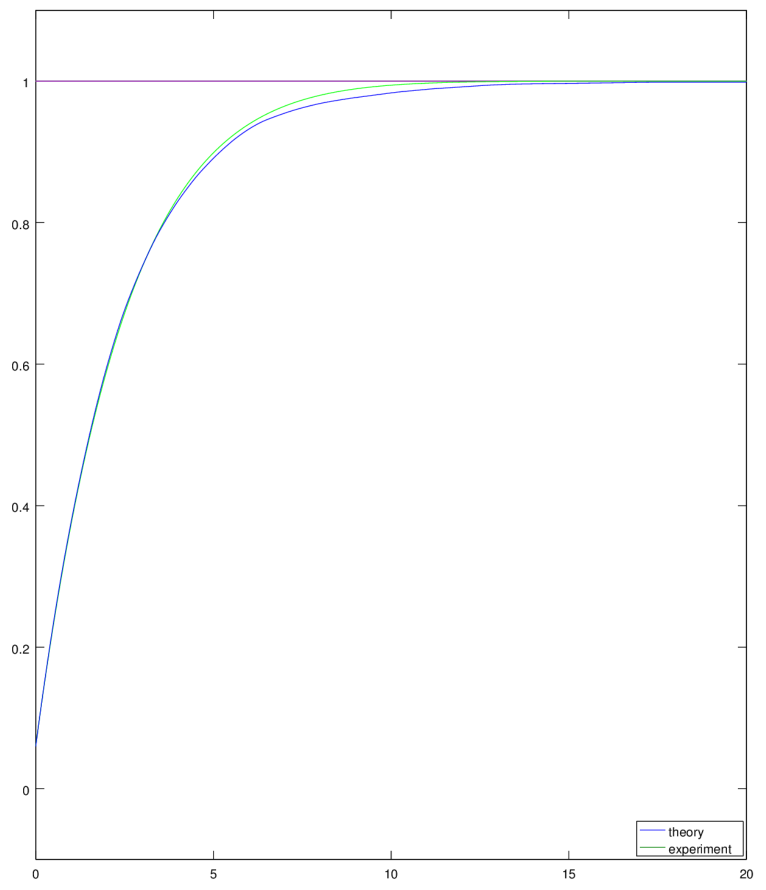


Figure 10: Expected values for $P(t)$. Here $A = 6.0, r = 0.1$, $\frac{r}{A} = 0.0166$.

4.4.5 Validating the Expected Number of Drones

We implemented a model which simulates the movement of drones within a 3 by 2 rectangle. The drones bounce off of the walls as described. We recorded the amount of time spent t or fewer minutes from the next observation, and by dividing by the total simulation time, converted that into an expected probability of observation within the next t minutes.

The model's predictions hewed to our predictions uncannily well. They are presented below:



Our model underestimates the probability, but only by a very small amount. The code for this images is presented in the code appendix.

4.4.6 How Many Drones to Buy?

Suppose the goal of a drone purchasing program is: achieve $f(t)$-percent coverage after t time units. With n drones, the chance of being observed after t time units is:

$$f(t) = 1 - \left(1 - \frac{\pi r^2 + 2r(vt)}{A}\right)^n \tag{7}$$

Therefore, solving for n:

$$n = \frac{\log\left(1 - f(t)\right)}{\log\left(1 - \frac{\pi r^2 + 2r(vt)}{A}\right)} \tag{8}$$

Earlier we determined the following parameter values:

Parameter	Value
A	$(4.828\text{km})^2$ or $23,309,584m^2$
r	$300m$
v	$16m/sec$

If we want to be observed every 15 minutes 95% of the time, this will require 6.2081 drones. We can round up to 7 drones.

4.4.7 Robustness Testing

Suppose v is smaller than it should be. This could happen for many reasons: there could be rain which impedes the motion of the drones, there could be a power shortage, there could be interfering regulations. We want to see how this change in v will affect our predictions.

In Figure 11, we show how the probability of having been observed after 15 minutes decays as the velocity decreases. The rounding-up to have an integral number of drones has helped, and the velocity can dip down to $15m/sec$ without disruptions.

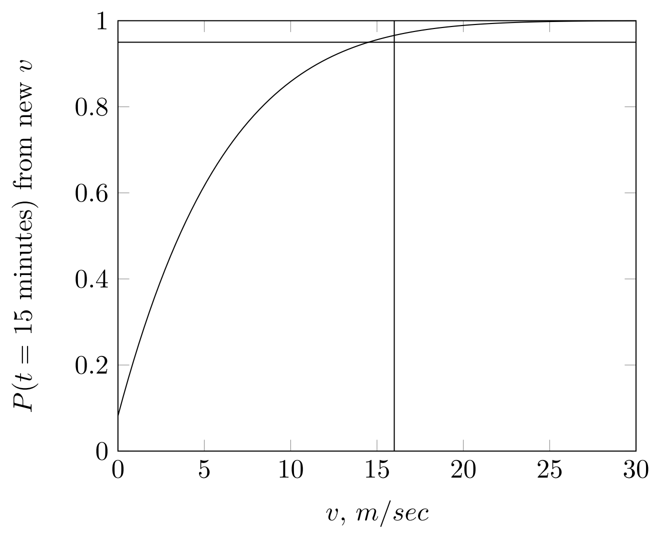


Figure 11: Robustness Test for Changed Velocity Values. If the velocity is above about $15m/sec$, then the desired 95% coverage is still achieved.

5 References

6 Math Appendix

In the fourth model, we mentioned that there was a rigerous mathematical derivation of the $\frac{\pi r^2+2r(vt)}{A}$ rule. We present it below:

We desire to find the probability that a point x_c is observed between times 0 and t. We assume that the drone is uniformly distributed throughout the domain, and that the boundary of A is further from x_c than $vt + r$, the largest distance away from x_c that a drone can start and still observe it by time t. We further assume that the drone velocity direction is uniformly distributed as well.

First, we will find the probability that a drone a distance a away from x_c is on a trajectory to observe x_c within t seconds. If $a < r$, then the drone is already observing x_c, so the probability is 1.

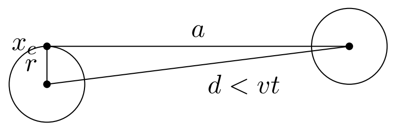


Figure 12: If the drone is closer than $\sqrt{(vt)^2 - r^2}$, then the drone will always make it in time. It only matters if it misses or not

7 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 = 10000; % number of events
n_drones = 10; % 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);

target = [1.0 1.0];


t = 0;
time_intervals = zeros(1,n_steps); % save event times
within_circle = norm(min(target   x)) < r
circle_status = zeros(1,n_steps);

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)));
```

```
[t_so_far, which_wall] = min([tl, tr, tb, tt]);

    a = v(:,1).^2 + v(:,2).^2;
    neg_b = 2*(v(:,1).*(target(1)  x(:,1)) + v(:,2).*(target(2)  x(:,2)));
    c = (x(:,1)  target(1)).^2 + (x(:,2)  target(2)).^2  r.^2;
    discriminant = neg_b.^2  4*a.*c;
    for j = 1:n_drones
        if discriminant(j) > 0
            t_enter = (neg_b(j)  sqrt(discriminant(j)))./(2*a(j));
            t_exit = (neg_b(j) + sqrt(discriminant(j)))./(2*a(j));
            if t_enter > eps && t_enter < t_so_far
                t_so_far = t_enter;
                within_circle = 1;
                which_wall = 0;
            elseif t_exit > eps && t_exit < t_so_far
                t_so_far = t_exit;
                within_circle = 0;
                which_wall = 0;
            end
        end
    end
    circle_status(i) = within_circle;

    % find the minimum overall time until a wall collision,
    %         and which wall will be collided with

    % move the drones forwards and advance time
    x += v * t_so_far;
```

```
t += t_so_far;
printf('%d',which_wall)

% 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

for j = 1:n_drones
        if (x(j,1) < r) || (x(j,1) > w r) || (x(j,2) < r) || (x(j,2) > h
                theta = rand();
                v(j,:) = [ cos(2*pi*theta) sin(2*pi*theta)];
                x(j,:) = [r,r] + [w r,h r].*rand(1,2);
        end
end

% record the timestamp and the drone positions at this time
time_intervals(i) = t_so_far;
```

```
        X(:,i,:) = x;
end

printf('\n')

max_time_until = 20.0
time_untils = (0.0:0.1:max_time_until);
n = length(time_untils);

t_max = sum(time_intervals);
time_in_observation = zeros(1,n);
last_i =  1;
for j = 1:n
        for i = 1:(n_steps 1)
                if circle_status(i) == 1
                        time_since_last_collision = sum(time_intervals(last_i+2:
                        time_in_observation(j) += time_intervals(i+1) + min([tim
                        last_i = i;
                end
        end
end

A = (w   2*r)*(h 2*r);
predicted_probability_in_observation_of_one = (pi*r.^2 + 2*r*1.0*time_untils) / 

printf('\n');
mean(time_intervals)
```

```
%         subplot(2,2,1)
%         plot(cumsum(time_intervals),circle_status)
%
%         subplot(2,2,2)
%         hold on
%         for i = 1:n_drones
%                   plot(X(i,:,1),X(i,:,2))
%         end
%         hold off
%
%         subplot(2,2,3)
%         for i = 1:n_drones
%                   scatter(X(i,:,1),X(i,:,2))
%         end
%
%         subplot(2,2,4)
%
hold on
plot(time_untils,ones(length(time_untils)))
plot(time_untils, 1.0   (1.0   predicted_probability_in_observation_of_one).^n_d
plot(time_untils, time_in_observation/t_max,'color','blue');
legend('theory','experiment','location','southeast');
axis([0,max_time_until, 0.1,1.1])
hold off

print('theory vs experiment.png')
pause
```

Listing 2: Basic Coverage

```
% We can define this problem as a discrete grid of points to traverse. We
% declare each grid point spaced by the distance traveled

% No geographic point in the city should remain unobserved from the air for
% more than 15 minutes in a row.

length = 4828;
width = 4828;
area = length*width;    % area of city

ts = 15*60/48;  % time step per change in i
t_visit = 15*60;

w = 304.8; % width of the observable range of quad copters
w2 = w*w; % square approx of area observed by drone

v = 16;  % velocity of the drone m/s
area_covered = v*15*60*304.8;   % area covered by drone after 15 minutes
drones_needed1 = ceil(area/area_covered); % drones needed for Method I
num_cov_temp = area_covered/w2; % number of boxes needed in grid per drone
num_cov = ceil(num_cov_temp);   % rounded up

num_boxes_temp = area/w2;    % number of boxes needed in grid
s_num = ceil(sqrt(num_boxes_temp));
num_boxes = s_num*s_num;    % approximate number to make even square

x = 1;
y = 1;
```

```
t = 0;

% Base case where we can assume the copter observes 1 block per time step
% (to scale)

% Initialize location array
location = zeros(s_num);

% Initialize time since visited array
visit = zeros(s_num);

for i=1:s_num
    for j=1:s_num
        visit(i,j) = t_visit;
    end
end

% Initialize offsets of drones
pos_2x = 6;
pos_2y = 0;
pos_3x = 10;
pos_3y = 0;
pos_4x = 0;
pos_4y = 8;
pos_5x = 6;
pos_5y = 8;
pos_6x = 10;
pos_6y = 8;
```

```
% Initialize position of drones
location(x,y) = 1;
location(x + pos_2x, y + pos_2y) = 1;
location(x + pos_3x, y + pos_3y) = 1;
location(x + pos_4x, y + pos_4y) = 1;
location(x + pos_5x, y + pos_5y) = 1;
location(x + pos_6x, y + pos_6y) = 1;

% Initialize movement array of each drone
d1x = [x];
d1y = [y];
d2x = [x + pos_2x];
d2y = [y + pos_2y];
d3x = [x + pos_3x];
d3y = [y + pos_3y];
d4x = [x + pos_4x];
d4y = [y + pos_4y];
d5x = [x + pos_5x];
d5y = [y + pos_5y];
d6x = [x + pos_6x];
d6y = [y + pos_6y];

%disp(location);

for j = 1:3
    for i = 1:48
```

```
if (min(min(visit)) == 0)
    disp('Failure to observe');
end

% Add time step
t = t + ts;

% Update time since visited matrix
visit = visit   ts;

if (i <= 5)

    x = x + 1;

    location(x,y) = 1;
    location(x 1 ,y) = 0;
    visit(x,y) = t_visit;

    location(x + pos_2x, y + pos_2y) = 1;
    location(x + pos_2x   1, y + pos_2y) = 0;
    visit(x + pos_2x, y + pos_2y) = t_visit;

    location(x + pos_3x, y + pos_3y) = 1;
    location(x + pos_3x   1, y + pos_3y) = 0;
    visit(x + pos_3x, y + pos_3y) = t_visit;

    location(x + pos_4x, y + pos_4y) = 1;
    location(x + pos_4x   1, y + pos_4y) = 0;
```

```
        visit(x + pos_4x, y + pos_4y) = t_visit;

        location(x + pos_5x, y + pos_5y) = 1;
        location(x + pos_5x   1, y + pos_5y) = 0;
        visit(x + pos_5x, y + pos_5y) = t_visit;

        location(x + pos_6x, y + pos_6y) = 1;
        location(x + pos_6x   1, y + pos_6y) = 0;
        visit(x + pos_6x, y + pos_6y) = t_visit;

        d1x = [d1x x];
        d1y = [d1y  y];
        d2x = [d2x  (x + pos_2x)];
        d2y = [d2y  (y + pos_2y)];
        d3x = [d3x  (x + pos_3x)];
        d3y = [d3y  (y + pos_3y)];
        d4x = [d4x  (x + pos_4x)];
        d4y = [d4y  (y + pos_4y)];
        d5x = [d5x  (x + pos_5x)];
        d5y = [d5y  (y + pos_5y)];
        d6x = [d6x  (x + pos_6x)];
        d6y = [d6y  (y + pos_6y)];

        %disp(location);
    end

    if (i >= 6 && i <=12)
```

```
y = y + 1;

location(x,y) = 1;
location(x,y 1) = 0;
visit(x,y) = t_visit;

location(x + pos_2x, y + pos_2y) = 1;
location(x + pos_2x, y + pos_2y   1) = 0;
visit(x + pos_2x, y + pos_2y) = t_visit;

location(x + pos_3x, y + pos_3y) = 1;
location(x + pos_3x, y + pos_3y 1) = 0;
visit(x + pos_3x, y + pos_3y) = t_visit;

location(x + pos_4x, y + pos_4y) = 1;
location(x + pos_4x, y + pos_4y 1) = 0;
visit(x + pos_4x, y + pos_4y) = t_visit;

location(x + pos_5x, y + pos_5y) = 1;
location(x + pos_5x, y + pos_5y   1) = 0;
visit(x + pos_5x, y + pos_5y) = t_visit;

location(x + pos_6x, y + pos_6y) = 1;
location(x + pos_6x, y + pos_6y 1) = 0;
visit(x + pos_6x, y + pos_6y) = t_visit;

d1x = [d1x x];
d1y = [d1y  y];
```

```
    d2x = [d2x  (x + pos_2x)];
    d2y = [d2y  (y + pos_2y)];
    d3x = [d3x  (x + pos_3x)];
    d3y = [d3y  (y + pos_3y)];
    d4x = [d4x  (x + pos_4x)];
    d4y = [d4y  (y + pos_4y)];
    d5x = [d5x  (x + pos_5x)];
    d5y = [d5y  (y + pos_5y)];
    d6x = [d6x  (x + pos_6x)];
    d6y = [d6y  (y + pos_6y)];

    %disp(location);
end

if ((i == 13) || (i==20) || (i == 27) || (i==34) || (i == 41))

    x = x   1;

    location(x,y) = 1;
    location(x+1,y) = 0;
    visit(x,y) = t_visit;

    location(x + pos_2x, y + pos_2y) = 1;
    location(x + pos_2x + 1, y + pos_2y) = 0;
    visit(x + pos_2x, y + pos_2y) = t_visit;

    location(x + pos_3x, y + pos_3y) = 1;
    location(x + pos_3x + 1, y + pos_3y) = 0;
```

```
visit(x + pos_3x, y + pos_3y) = t_visit;

location(x + pos_4x, y + pos_4y) = 1;
location(x + pos_4x + 1, y + pos_4y) = 0;
visit(x + pos_4x, y + pos_4y) = t_visit;

location(x + pos_5x, y + pos_5y) = 1;
location(x + pos_5x + 1, y + pos_5y) = 0;
visit(x + pos_5x, y + pos_5y) = t_visit;

location(x + pos_6x, y + pos_6y) = 1;
location(x + pos_6x + 1, y + pos_6y) = 0;
visit(x + pos_6x, y + pos_6y) = t_visit;

d1x = [d1x x];
d1y = [d1y  y];
d2x = [d2x  (x + pos_2x)];
d2y = [d2y  (y + pos_2y)];
d3x = [d3x  (x + pos_3x)];
d3y = [d3y  (y + pos_3y)];
d4x = [d4x  (x + pos_4x)];
d4y = [d4y  (y + pos_4y)];
d5x = [d5x  (x + pos_5x)];
d5y = [d5y  (y + pos_5y)];
d6x = [d6x  (x + pos_6x)];
d6y = [d6y  (y + pos_6y)];

%disp(location);
```

```
end

if ((i >=14 && i <=19) || (i >= 28 && i <= 33) || (i>=42))

    y = y   1;

    location(x,y) = 1;
    location(x,y+1) = 0;
    visit(x,y) = t_visit;

    location(x + pos_2x, y + pos_2y) = 1;
    location(x + pos_2x, y + pos_2y + 1) = 0;
    visit(x + pos_2x, y + pos_2y) = t_visit;

    location(x + pos_3x, y + pos_3y) = 1;
    location(x + pos_3x , y + pos_3y+ 1) = 0;
    visit(x + pos_3x, y + pos_3y) = t_visit;

    location(x + pos_4x, y + pos_4y) = 1;
    location(x + pos_4x, y + pos_4y + 1) = 0;
    visit(x + pos_4x, y + pos_4y) = t_visit;

    location(x + pos_5x, y + pos_5y) = 1;
    location(x + pos_5x, y + pos_5y + 1) = 0;
    visit(x + pos_5x, y + pos_5y) = t_visit;

    location(x + pos_6x, y + pos_6y) = 1;
    location(x + pos_6x, y + pos_6y + 1) = 0;
    visit(x + pos_6x, y + pos_6y) = t_visit;

    d1x = [d1x x];
    d1y = [d1y  y];
    d2x = [d2x  (x + pos_2x)];
    d2y = [d2y  (y + pos_2y)];
    d3x = [d3x  (x + pos_3x)];
    d3y = [d3y  (y + pos_3y)];
    d4x = [d4x  (x + pos_4x)];
    d4y = [d4y  (y + pos_4y)];
    d5x = [d5x  (x + pos_5x)];
    d5y = [d5y  (y + pos_5y)];
    d6x = [d6x  (x + pos_6x)];
    d6y = [d6y  (y + pos_6y)];

    %disp(location);
end

if ((i >=21 && i <=26) || (i >= 35 && i <= 40))

    y = y + 1;

    location(x,y) = 1;
    location(x,y 1) = 0;
    visit(x,y) = t_visit;

    location(x + pos_2x, y + pos_2y) = 1;
    location(x + pos_2x, y + pos_2y   1) = 0;
```

```
visit(x + pos_2x, y + pos_2y) = t_visit;

location(x + pos_3x, y + pos_3y) = 1;
location(x + pos_3x , y + pos_3y   1) = 0;
visit(x + pos_3x, y + pos_3y) = t_visit;

location(x + pos_4x, y + pos_4y) = 1;
location(x + pos_4x, y + pos_4y   1) = 0;
visit(x + pos_4x, y + pos_4y) = t_visit;

location(x + pos_5x, y + pos_5y) = 1;
location(x + pos_5x, y + pos_5y   1) = 0;
visit(x + pos_5x, y + pos_5y) = t_visit;

location(x + pos_6x, y + pos_6y) = 1;
location(x + pos_6x, y + pos_6y   1) = 0;
visit(x + pos_6x, y + pos_6y) = t_visit;

d1x = [d1x x];
d1y = [d1y  y];
d2x = [d2x  (x + pos_2x)];
d2y = [d2y  (y + pos_2y)];
d3x = [d3x  (x + pos_3x)];
d3y = [d3y  (y + pos_3y)];
d4x = [d4x  (x + pos_4x)];
d4y = [d4y  (y + pos_4y)];
d5x = [d5x  (x + pos_5x)];
d5y = [d5y  (y + pos_5y)];
```

```
                d6x = [d6x  (x + pos_6x)];
                d6y = [d6y  (y + pos_6y)];

                %disp(location);
            end
        end
    end

    % disp(location);
    % disp(visit);

    % Plots current location of drones
    figure
    hold on;
    dd1 = plot(x,y);
    dd2 = plot(x + pos_2x, x + pos_2y);
    dd3 = plot(x + pos_3x, x + pos_3y);
    dd4 = plot(x + pos_4x, x + pos_4y);
    dd5 = plot(x + pos_5x, x + pos_5y);
    dd6 = plot(x + pos_6x, x + pos_6y);
    plot(d1x, d1y, 'b')
    plot(d2x, d2y, 'b')
    plot(d3x, d3y, 'b')
    plot(d4x, d4y, 'b')
    plot(d5x, d5y, 'b')
    plot(d6x, d6y, 'b')
    set(dd1,'Marker','square')
    set(dd2,'Marker','square')
```

```
set(dd3,'Marker','square')
set(dd4,'Marker','square')
set(dd5,'Marker','square')
set(dd6,'Marker','square')
t
```